Lumberyard
Legacy Reference
# Table of Contents

Amazon Lumberyard Legacy Reference ................................................................. 1

Animation System ........................................................................................................... 2

Animation Overview ........................................................................................................ 2

Linear Animations ........................................................................................................ 2

Interactive Animations ................................................................................................. 3

Scripted Animations .................................................................................................... 4

Character Asset Files ................................................................................................... 5

Character File (*.chr) ................................................................................................ 5

Character Definition File (*.cdf) ................................................................................ 5

Character Skinned Render Mesh (*.skin) ................................................................ 5

Maya Export Tools ...................................................................................................... 5

Accessing Maya Export Tools .................................................................................. 6

Setting Time Working Units for Maya ....................................................................... 8

Geometry Validation .................................................................................................. 8

Exporting Static Meshes ........................................................................................... 8

Exporting Multiple Meshes with a Collision Object ................................................ 10

Exporting Characters ................................................................................................. 12

Exporting Materials .................................................................................................. 15

Exporting Animations ................................................................................................. 16

Exporting Blendshapes ............................................................................................. 17

Exporting a Level of Detail (LOD) ............................................................................. 17

Exporting an Alembic Cache ...................................................................................... 24

Setting Export Options ............................................................................................. 25

3ds Max Export Tools ................................................................................................. 26

Exporting Static Meshes and Characters ................................................................ 26

Exporting Materials .................................................................................................. 28

Exporting Bones and Animations ............................................................................ 28

Exporting a Level of Detail (LOD) ............................................................................. 31

Configuring Log Options .......................................................................................... 32

Using Geppetto ......................................................................................................... 33

Geppetto Display Options ......................................................................................... 36

Creating a Character Definition .............................................................................. 40

Character Attachments .............................................................................................. 41

Animating Characters ............................................................................................... 60

Mannequin System ...................................................................................................... 82

Mannequin System Files ........................................................................................... 82

Creating a Mannequin Entity ..................................................................................... 85

Using Mannequin Editor ......................................................................................... 86

Using the New Mannequin Setup Window ............................................................... 111

Synchronizing Multiple Characters .......................................................................... 115

Using Flow Graph with Mannequin ......................................................................... 116

Debugging Mannequin System Issues ................................................................... 116

Physicalizing Characters (Ragdoll) ......................................................................... 117

Ragdoll Best Practices ............................................................................................... 117

Ragdoll Skeleton DCC Setup ................................................................................... 117

Creating Joint Mesh Proxies .................................................................................... 118

Using physParentFrames ........................................................................................ 121

Applying Simulation Settings to Ragdoll Joints ....................................................... 122

Lumberyard Proxy Tool (Experimental) ................................................................. 125

Adding Mesh Proxy Materials .................................................................................. 126

Ragdoll Physics ........................................................................................................ 126

Using Inverse Kinematics (IK) ................................................................................ 128

Aim IK (Aim Poses) .................................................................................................. 128

Look IK (Look Poses) .............................................................................................. 142
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud Canvas Flow Graph Node Reference</td>
<td>695</td>
</tr>
<tr>
<td>Placing Cached Shadows</td>
<td>693</td>
</tr>
<tr>
<td>Managing Flow Graph Modules</td>
<td>690</td>
</tr>
<tr>
<td>Using Flow Graph Links</td>
<td>688</td>
</tr>
<tr>
<td>Using Flow Graph Tokens</td>
<td>689</td>
</tr>
<tr>
<td>JSON Nodes</td>
<td>389</td>
</tr>
<tr>
<td>Kinect Nodes</td>
<td>392</td>
</tr>
<tr>
<td>Logic Nodes</td>
<td>393</td>
</tr>
<tr>
<td>Material Nodes</td>
<td>404</td>
</tr>
<tr>
<td>MaterialFX Nodes</td>
<td>407</td>
</tr>
<tr>
<td>Math Nodes</td>
<td>409</td>
</tr>
<tr>
<td>Mission Nodes</td>
<td>427</td>
</tr>
<tr>
<td>Module Nodes</td>
<td>432</td>
</tr>
<tr>
<td>Movement Nodes</td>
<td>436</td>
</tr>
<tr>
<td>Physics Nodes</td>
<td>439</td>
</tr>
<tr>
<td>Prefab Nodes</td>
<td>445</td>
</tr>
<tr>
<td>ProceduralMaterial Nodes</td>
<td>446</td>
</tr>
<tr>
<td>Related Console Variables</td>
<td>694</td>
</tr>
<tr>
<td>Recommended Settings</td>
<td>694</td>
</tr>
<tr>
<td>S3 (Storage) Nodes</td>
<td>708</td>
</tr>
<tr>
<td>SNS (Notification Service) Nodes</td>
<td>710</td>
</tr>
<tr>
<td>SQS (Message Queuing Service) Nodes</td>
<td>713</td>
</tr>
<tr>
<td>System Nodes</td>
<td>463</td>
</tr>
<tr>
<td>Time Nodes</td>
<td>465</td>
</tr>
<tr>
<td>Twitch Nodes</td>
<td>474</td>
</tr>
<tr>
<td>UI Flow Graph Nodes</td>
<td>482</td>
</tr>
<tr>
<td>Vec3 Nodes</td>
<td>646</td>
</tr>
<tr>
<td>Vehicle Nodes</td>
<td>654</td>
</tr>
<tr>
<td>Video Nodes</td>
<td>664</td>
</tr>
<tr>
<td>VideoPlayback Nodes</td>
<td>665</td>
</tr>
<tr>
<td>Virtual Reality (VR) Nodes</td>
<td>668</td>
</tr>
<tr>
<td>Weapon Nodes</td>
<td>672</td>
</tr>
<tr>
<td>XML Nodes</td>
<td>675</td>
</tr>
<tr>
<td>Using Console Variables</td>
<td>693</td>
</tr>
<tr>
<td>Using Flow Graph Debugger</td>
<td>692</td>
</tr>
<tr>
<td>Using Flow Graph Debugger</td>
<td>692</td>
</tr>
<tr>
<td>Using Logic Nodes</td>
<td>393</td>
</tr>
<tr>
<td>Using MaterialFX Nodes</td>
<td>407</td>
</tr>
<tr>
<td>Tutorial Walk-through</td>
<td>374</td>
</tr>
<tr>
<td>Using Prefab Nodes</td>
<td>445</td>
</tr>
<tr>
<td>Using ProceduralMaterial Nodes</td>
<td>446</td>
</tr>
<tr>
<td>Using Related Console Variables</td>
<td>694</td>
</tr>
<tr>
<td>Using Recommended Settings</td>
<td>694</td>
</tr>
<tr>
<td>Using Related Console Variables</td>
<td>694</td>
</tr>
</tbody>
</table>
Using the Designer Tool ................................................................. 967
  Designer Tool Settings ................................................................. 968
  Selection Tools ........................................................................... 969
  Shape Tools ................................................................................ 970
  Edit Tools .................................................................................. 972
  Modify Tools ............................................................................... 974
  Texture Tools ............................................................................... 976
  Miscellaneous Tools ................................................................. 978
Using the Measurement System Tool ............................................. 980
Using the Object Selector .............................................................. 980
  Finding an Object .................................................................... 981
  Object Selector Table ............................................................... 983
Brushes ........................................................................................ 983
  DrawLast .................................................................................. 985
Prefabs ........................................................................................ 985
Common Parameters and Properties ............................................. 986
  Entity Properties ....................................................................... 986
  Entity Parameters ..................................................................... 986
  Scripting and Flow Graph Entity Parameters .............................. 987
  Entity Links ............................................................................ 987
  Entity Events .......................................................................... 988
  Attached Entities ..................................................................... 988
  Shape Parameters .................................................................... 989
Entity Reference .......................................................................... 989
  Actor Entity ........................................................................... 990
  AI Control Objects .................................................................. 990
  Anim Entities .......................................................................... 995
  Archetype Entity ...................................................................... 995
  Area Entities .......................................................................... 996
  Audio Entities .......................................................................... 1002
  Boid Entity ............................................................................ 1006
  Camera Entity .......................................................................... 1009
  Geom Entities .......................................................................... 1010
  Light Entities .......................................................................... 1011
  Lightning Arc Entity .................................................................. 1015
  Miscellaneous Entities ........................................................... 1017
  Particle Entities ....................................................................... 1019
  Physics Entities ......................................................................... 1020
  Rain Entity .............................................................................. 1029
  Render Entities ......................................................................... 1030
  River Entity ............................................................................ 1030
  Road Entity .............................................................................. 1031
  Rope Entity ............................................................................. 1031
  Snow Entity ............................................................................ 1034
  Tornado Entity ......................................................................... 1035
  Trigger Entities .......................................................................... 1035
Entity Property Prefixes ................................................................. 1037
Creating a New Entity Class ......................................................... 1038
Entity Pool System ......................................................................... 1039
  Editor Usage ........................................................................... 1040
  Static versus Dynamic Entities ............................................... 1040
  Entity Pool Definitions ............................................................ 1040
  Entity Pool Creation ............................................................... 1042
  Creating and Destroying Static Entities with Pools .................... 1043
  Creating and Destroying Dynamic Entities with Pools .............. 1045
  Serialization ........................................................................... 1046
  Listener/Event Registration ....................................................... 1047
Use the Amazon Lumberyard Legacy Reference for information regarding legacy systems and features, such as Flow Graph.

- **Amazon Lumberyard User Guide** – Learn more about the most recent features and systems for Lumberyard.
- **Amazon Lumberyard Getting Started Guide** – Get familiar with Lumberyard basics, such as navigating the editor, building terrain, and lighting a scene.
- **Amazon Lumberyard C++ API Reference** – Learn more about the fundamental C++ API operations of the Lumberyard component entity system.
- **Amazon Lumberyard Release Notes** – Learn more about improvements, highlights, and known issues for Lumberyard features and systems.
Animation System

The legacy character animation system combines skeletal-based deformation of meshes with morph-based vertex deformation to allow for complex animation. Character movements appear much more realistic by playing and blending animation sequences, controlling facial expressions, and applying damage effects. Characters can play scripted movements, employ AI navigation, or use the Mannequin system to play complex, fully interactive animation sequences, either alone or in concert with other characters.

The recommended animation frame rate is 30 fps. If you are creating animations in Maya, there are additional supported frame rates of 15 fps, 60 fps, 120 fps, and 240 fps.

Topics
- Animation Overview (p. 2)
- Character Asset Files (p. 5)
- Maya Export Tools (p. 5)
- 3ds Max Export Tools (p. 26)
- Using Geppetto (p. 33)
- Mannequin System (p. 82)
- Physicalizing Characters (Ragdoll) (p. 117)
- Using Inverse Kinematics (IK) (p. 128)
- Animation Events (p. 148)
- Limb IK Technical (p. 148)
- Animation Streaming (p. 149)
- Animation Debugging (p. 152)
- Fall and Play (p. 156)
- Time in the Animation System (p. 157)

Animation Overview

One of Lumberyard's goals is to push the boundaries of animations, which are all rendered in real time. Lumberyard provides tools to create both linear and interactive animations:

- Linear animation is the kind of animation seen in movies and cut-scenes, which play as a video.
- Interactive animation is used to convey AI and avatar (player) behavior, with sequences dependent on player choices in gameplay.

There is a big difference between how each type of animation is incorporated into a game, although this difference may not be obvious to the player, who simply sees characters moving on-screen. The key difference is in the decision-making process: who decides what a character on the screen is going to do next?

Linear Animations

In linear animation, the decision-making process happens inside the head of the people designing the animation. During this process, an animator has direct control over every single keyframe. They don't need to deal with collision detection, physics and pathfinding; characters only run into walls or collide with each other when the animator wants them to. AI behavior does not need to react to player
behavior; the person who writes the storyboard decides how intelligent or stupid the characters are. To show interactions between characters, you can put them in motion-capture suits and record their performances.

A linear animation sequence needs to show action from a single camera angle because the audience won't be moving during the animation; as a result, animators don't need to deal with transitions and motion combinations; they control every aspect of the motion clip. Because everything is fixed and predictable, it's possible to guarantee a consistent motion quality. Animators can always go back and adjust details in the scene, such as add or delete keyframes, adjust the lighting, or change the camera position.

The technical challenges with creating linear animation primarily involve rendering issues, such as not dropping the frame rate and ensuring that facial and body animations are in sync.

All linear animations in Lumberyard are created with Track View editor.

Interactive Animations

Creating interactive animations presents significantly tougher challenges. Animators and programmers do not have direct control over a character's on-screen movements. It is not always obvious where and how the decision-making process happens. It is usually a complex combination of AI systems, player input, and sometimes contextual behavior.

By definition, interactive animation is responsive. It looks visibly different depending on an individual user's input and adapts automatically to actions on the screen. Moving from linear animation to interactive animation requires more than just a set of small tweaks or a change in complexity—it requires a completely different technology under the hood. With interactive animation, an animator cannot precisely plan and model a character's behavior. Instead, animators and programmers develop a system that allows them to synthesize motion automatically and define rules for character behavior.

Automatic motion synthesis is a crucial feature in making animation more interactive. A system that synthesizes motion must be very flexible, because it is difficult to predict the sequence of actions that a character may take, and each action can start at any time.

Imagine, for example, a character moving through an outdoor environment. At a minimum, the designer needs to specify the style, speed, and direction of the character's locomotion. There should also be variations in motion while running uphill or downhill, leaning when running around corners or carrying objects of different sizes and weights—the character should run faster while carrying a pistol than when hefting a rocket launcher. It might also be necessary to interactively control emotional features such as happiness, anger, fear, and tiredness. Additionally, the character may need to perform multiple tasks simultaneously, such as walking in one direction, turning head and eyes to track a bird in another direction, and aiming a gun at a moving object in third direction. Providing unique animation assets for every possible combination and degree of freedom is nearly impossible and would involve an incredibly large amount of data. A mechanism for motion modifications is needed to keep the asset count as low as possible.

Developing such a system involves close collaboration and a tight feedback loop between programmers, animators, and designers. Problems with the behavior and locomotion systems (either responsiveness or motion quality) are usually addressed from several sides.

Interactive animation can be divided into two categories: Avatar control and AI control. In both cases, animators and programmers have indirect control over the actual behavior of a character in gameplay, because decision making for the character's next action happens elsewhere. Let's take a closer look at the situation in game environments.

Avatar control

An avatar character is controlled by the game player, whose decisions determine all of the avatar's actions. The locomotion system takes the player's input and translates it on the fly into skeleton
movements (using procedural and data-driven methods). With avatar control, high responsiveness is the top priority, while motion quality might be limited by the game rules. This means that many well-established rules for ‘nice’-looking animations are in direct conflict with the responsiveness you need for certain types of gameplay.

The quality of animations as executed on the screen depends largely on the skills and decisions of each player controlling the character—they decide what the avatar will do next. Because a player's actions are unpredictable, motion planning based on predictions is not possible. Complex emotional control is not possible (and probably not needed). It's only possible on a raw level, such as soft punch versus an aggressive punch. However, it might be possible to let the player control the locomotion of the avatar, and to let the game code control the emotional behavior of the avatar by blending in “additive animations” based on the in-game situation.

In all these scenes, the player is controlling the character with a game pad. The character's presentation on the screen is using animation assets created by animators.

**AI control**

For AI characters, the decision-making process happens entirely inside the game code. Game developers design a system to generate behavior, which acts as an intermediary between the game creators and players. For the system to perform this task, it is necessary for game designers to explicitly specify behavioral decisions and parameters for AI characters, including a clear definition of the rules of movements for each character type. Interactive animation for AI characters is much harder to accomplish than animations for avatars, but at the same time it offers some (not always obvious) opportunities to improve motion quality. High responsiveness is still the primary goal but, because character choices happen inside the game code, it is possible in certain circumstances to predict a character's actions. If the AI system knows what the AI character wants to do next, then it is possible to incorporate this knowledge into motion planning. With good motion planning, interactive animation might be able to use more classical or ‘nice’ animation rules. As a result, AI control can have a somewhat higher motion quality than avatar control, though at the cost of having more complex technology under the hood.

The only source of uncertainty in such a prediction system is the player: the AI reacts to the player, and predicting the player's actions is impossible. As a result, it's nearly impossible to create the right assets for every in-game situation, and this in turn makes it impossible to guarantee a consistent motion quality. For an animator working on interactive animation, it can be a significant problem to have no direct control over the final animation—it's never clear when the work is complete. This is one reason why the linear animation in movies and cut-scenes look superior, and why interactive animations can be troublesome.

Lumberyard tackles the problem with interactive animation in multiple levels:

- In the low-level CryAnimation system library, the engine provides support for animation clips, parametrized animation, and procedural modification of poses. Animations can be sequenced together or layered on top of each other in a layered transition queue.
- In the high-level CryAction library, the CryMannequin system helps to manage the complexity of animation variations, transitions between animations, animations that are built up out of many others, sequencing of procedural code, links to game code, and so on.

**Scripted Animations**

Because interactive animation is much more difficult than linear animation, many games blur the line between cut-scenes and in-game actions by using interactive scripted sequences.

In this case, characters act on a predefined path. The quality of this kind of motion can be very high. Because it is not fully interactive, animators have more control over the entire sequence, a kind of
manually designed motion planning. These are perfectly reasonable cheats to overcome hard-to-solve animation problems. It may be even possible to script the entire AI sequence to allow near-cut-scene quality. The action feels interactive and looks absolutely cinematic, but it is actually more an illusion of interactivity.

In the game Crysis, Crytek designers made use of scripted animations in many scenes. In the "Sphere" cut-scene, the Hunter is shown walking uphill and downhill and stepping over obstacles. This is a scripted sequence where the assets were made for walking on flat ground, but Crytek used CCD-IK to adapt the character's legs to the uneven terrain. In the "Fleet" cut-scene with the Hunter on the carrier deck, the player can move around while the Hunter is fighting other non-playing characters.

Both scenes look and feel highly interactive but they are not. The Hunter doesn't respond to the player and the player cannot fight the Hunter. The scenes are fully linear and scripted, basically just animated background graphics. These sequences were created in Track View editor. Some of them used the Flow Graph Editor. When the cut-scene is over, the Hunter turns into an AI-controlled interactive character.

Character Asset Files

You can export the following character file types for use in Lumberyard.

**Character File (*.chr)**

You create the .chr file in a DCC tool. This file contains the base skeleton.

**Character Definition File (*.cdf)**

You create the .cdf file in Geppetto. This file contains the base character, plus all attachments.

**Character Skinned Render Mesh (*.skin)**

You create the .skin file in a DCC tool. This file contains skinned character data. This data can be any asset that is animated with bone-weighted vertices, such as humans, aliens, ropes, lamps, heads, and parachutes. The .skin file includes the mesh, vertex weighting, vertex colors, and morph targets.

Maya Export Tools

Lumberyard Tools is a plugin for Autodesk Maya 2014, 2015, 2016, and 2017 that exports geometry, animated geometry, skinned geometry, and skeletons (joint hierarchies) from Maya into Lumberyard.

**To install the Lumberyard Tools plugin**

1. Navigate to the Lumberyard root directory (\lumberyard\dev) and run Lumberyard Setup Assistant.
2. On the **Install plugins** page, install Autodesk Maya.

**Topics**

- Accessing Maya Export Tools (p. 6)
- Setting Time Working Units for Maya (p. 8)
Accessing Maya Export Tools

To install this plugin, run SetupAssistant.bat. On the Install Plugins page, install Autodesk Maya. After it is installed, the Lumberyard tab is available in the user interface of Maya. This tab presents a series of options, including the Lumberyard Tools beaver icon.

After the Lumberyard Tools dialog box opens, the following is shown:
Setting Time Working Units for Maya

We recommend that you use the NTSC (30 fps) setting for animations, but there are additional supported frame rates of 15 fps, 60 fps, 120 fps, and 240 fps.

To change time working units to NTSC

1. In Maya, choose Window, Settings/Preferences, Preferences.
2. In the Preferences dialog box, choose Settings.
3. Under Working Units, we recommend that you choose NTSC (30fps), but NTSC Field (60fps) and Film (240fps) are also valid options.
4. Choose Save.

Geometry Validation

Before export, the plugin validates your character geometry. Be sure to resolve any errors that are displayed in the Lumberyard Validation window. For each error listed, choose Focus for more information about the error, as displayed in the Transform Attributes panel of the Attribute Editor for Maya. Errors are displayed on red backgrounds and warnings are on yellow backgrounds.

Exporting Static Meshes

To export static geometry, do the following steps. Make sure you save your scene before you export geometry.
To export static geometry

1. In Maya, choose the Lumberyard tab, and then choose the Lumberyard Tools beaver icon.
2. Select a geometry or group node in Maya, then choose **Add Selected** to add the node or group to the **Geometry Export** list window. You can only add one node or group (can be a group with children groups also) to the export list at a time. Select the check box to add the node or group for export. Choose the X to remove the node from the **Geometry Export** list. Choose the node name to edit the text as needed.

   **Note**
   Choose **Select** to see the node in Maya that corresponds to the export node in the **Geometry Export** list.

3. In the drop-down list, select **Geometry (.CGF)**.
4. For **Export Path**, choose the folder icon and select a directory path. By default, this path is the same as the directory of the current Maya file and all nodes are exported to this directory. Choose **Explorer View** to view the directory.
5. Expand **Advanced Options** and choose the following options as needed for the export node you selected:

   - **Merge Nodes** – Compiles geometry from multiple nodes into a single flattened mesh. Only supported for non-skinned geometry.
   - **8 Weights (skin only)** – Exports up to eight weights per skinned vertex. Generally used for faces or blend shapes.
   - **Vertex Colors** – Exports vertex colors.
   - **32 Bit Vertex** – Enabling this will add 32-bits of precision to position each vertex accurately when the mesh is located far from its pivot.

   When working in centimeter units, 32-bit vertex precision is useful when the geometry is more than 10 meters from the pivot. When working in meter units, 32-bit vertex precision is useful when geometry is more than 100 meters from the pivot.

   - For **Custom Path**, choose the folder icon and select a specific file path for your geometry. You can save each geometry to an individual location. This path overwrites the **Export Path** from the previous step.
6. Repeat as needed for each node you want to export. Make sure the check box is selected for each node you wish to export; otherwise that node will not be exported.
7. With the desired node or group selected in the Maya scene, in the **Material Export** section, choose **Add Group**. This creates a material group and adds all of the materials that were applied to the mesh.
8. When finished, choose **Export Geometry**.

**Exporting Multiple Meshes with a Collision Object**

You can use the Lumberyard Tools for Maya and the **FBX Settings** tool to generate your meshes. In Maya, create your material and set the collision property so that Lumberyard can identify the mesh as a collision.

For more information about the .fbx format, see **FBX Export** in the Autodesk Maya LT documentation.

**To export a mesh group with a collision object**

1. In Autodesk Maya, create your objects and define your collision object.
2. On the Maya tool shelf, click **Lumberyard Tools**.
3. Click the beaver icon to open the **Lumberyard Tools** window.
4. In the **Asset Browser** pane, select your objects and press **Ctrl+G** to group the objects.
5. In the outliner or channel box, name the new group. We recommend appending the group name with _group.
6. In the outliner or channel box, select the group name.
7. In the Lumberyard Tools window, under Geometry Export, click Add Selected. The group name appears under Geometry Export.

![Geometry Export](image1.png)

8. Before you can export an object, you must assign a material to the object. To do so, right-click the object in the viewport and choose Assign New Material.
9. In the Assign New Material dialog box, select a Phong shader.
10. In the viewport, select the group node with the newly assigned material.
11. In the Lumberyard Tools window, under Material Export, click Add Group. The default group name (from the first material) appears under Material Export.

![Material Export](image2.png)

12. (Optional) Under Material Export, rename the material for the collision object to be called collision.
13. Under Material Export, choose Proxy No Draw from the list for the collision shader type. You can also update this shader type later in the Lumberyard Material Editor.
14. Save the file to your project directory.
15. Click Export All. Be sure to save to the directory where your Maya file types are saved.

To validate the collision box in Lumberyard Editor

1. Open Lumberyard Editor.
2. In the Asset Browser pane, locate the object that you created.
3. Drag the object into the viewport.
4. To frame the object, press Z.
5. Open the Console Variables window and search for p_draw_helpers.
6. To show the collision path, set p_draw_helpers to 1. You should now see your collision box in the viewport.

Exporting Characters

Before you can use the Lumberyard Tools plugin for exporting character geometry from Maya, you must check the Up Axis setting under World Coordinate System for your scene. By default, this setting is Y for Maya. To check this setting, click Windows, Settings, Preferences. In the Preferences window, under Categories, click Settings.
If the **Up Axis** is set to Y, you must ensure the following is true:

- The root joint of the character is positioned at the origin of the scene at 0, 0, 0.
- The root joint of the character is oriented to **z-up** and **y-forward**.
- The **Joint Orient** attribute for the root joint is set to -90, 180, 0.
- A **SceneRoot** node exists for your scene. If this node does not exist, create it by choosing **Tools, Add Scene Root**.

If the **Up Axis** is set to Z, you must ensure the following is true:

- The root joint of the character is positioned at the origin of the scene at 0, 0, 0.
- The root joint of the character is oriented to **z-up** and **y-forward**.
- The **Joint Orient** attribute for the root joint is set to 0, 0, 0.
• A **SceneRoot** node does not exist for your scene.

The following procedure is very similar to the procedure on exporting static geometry, with many of the same options and advanced options. Refer to the previous procedure for explanation.
To export character geometry

1. In Maya, select the root joint node of the character.
2. In Lumberyard Tools, select the root joint node, then choose Add Selected. Be sure that it is set to Skeleton (.CHR) in the drop-down list.
3. Select the geometry that is skinned to the joints and choose Add Selected. Be sure that it is set to Skin (.SKIN) in the drop-down list.
4. Add the skinned geometry's materials to the Material Export list.
5. Choose Export Geometry.

Exporting Materials

There a couple of ways to export material (.mtl) files. All exported materials must be contained in a material group as shown in the following. Be sure to save your scene before you export your materials.

Lumberyard also uses material information to drive physics properties.

To export character materials (Method 1)

1. In your Maya scene, choose the geometry that has the desired materials applied.
2. In Lumberyard Tools in the Material Export section, choose Add Group. This creates a new material group and automatically adds all applied materials to it.
3. In the No Physics, choose from the following options:
   - No Physics - Material contains no physics attributes (default setting).
   - Default - Render geometry is used as a physics proxy. This is expensive for complex objects, so use this only for simple objects like cubes or if you need to fully physicalize an object.
   - ProxyNoDraw - Mesh is used exclusively for collision detection and is not rendered.
• **No Collide** - Proxy is used to detect player interaction, such as for vegetation touch bending.
• **Obstruct** - Used for "Soft Cover" to block AI agent views, such as for dense foliage.

4. For **Export Path**, choose the folder icon and select a directory path. By default, this path is the same as the directory of the current Maya file, and all nodes will be exported to this directory. To export to a custom directory, choose **Advanced Options, Custom Path**, choose the folder icon, and select a specific file path for your materials. You can save each material to an individual location. This path overwrites the **Export Path** from the previous step.

5. Make sure the check box for each material you wish to export is selected, then choose **Export Materials**.

### To export character materials (Method 2)

1. With nothing selected in the Maya scene, in Lumberyard Tools, choose **Add Group** to create an empty material group.
2. Select the newly created material group. Only material groups that are selected are exported. Choose the X to remove a material group as needed.
3. In Maya, select the materials in the **Hypershade** window you wish to add to this material group. Alternatively, you can select meshes that have the desired materials applied.

   **Note**
   Use the **Hypershade** button in Lumberyard Tools to display the material or group in the Maya **Hypershade** window for a selected material in the Lumberyard Tools **Material Export** window.

4. Choose **Add Material**.
5. In **No Physics**, choose from the following options:
   • **No Physics** - Material contains no physics attributes (default setting).
   • **Default** - Render geometry is used as a physics proxy. This is expensive for complex objects, so use this only for simple objects like cubes or if you need to fully physicalize an object.
   • **ProxyNoDraw** - Mesh is used exclusively for collision detection and is not rendered.
   • **No Collide** - Proxy is used to detect player interaction, such as for vegetation touch bending.
   • **Obstruct** - Used for "Soft Cover" to block AI agent views, such as for dense foliage.

6. For **Export Path**, choose the folder icon and select a directory path. By default, this path is the same as the directory of the current Maya file, and all nodes will be exported to this directory. If you want to export to a custom directory, choose **Advanced Options, Custom Path**, choose the folder icon, and select a specific file path for your materials. You can save each material to an individual location. This path overwrites the **Export Path** from the previous step.

7. Make sure the check box for each material you wish to export is selected, then choose **Export Materials**.

**Tip**
The order of materials listed can be changed by clicking on a material with the middle mouse button and dragging the material to the desired placement within the material group. This does not allow you to move a material to a different material group, however.

### Exporting Animations

Lumberyard Tools uses the Lumberyard Tools Animation Manager to specify various settings for each animation you want to export. New fields added to Lumberyard Tools Animation Manager also update the **Animation Export** window.

Be sure to save your Maya scene before you export animation.
Animation layers can be used to toggle animation keyframes on a node. By default all animations are on a BaseAnimation layer. If new animation layers are added to a Maya file, they are reflected in the Lumberyard Layers drop-down list in Lumberyard Animation Manager. If an animation layer is selected, keyframes on the animation layer will be exported. If an animation layer is not selected, the keyframes on those layers will not be exported.

To export character animations

1. In Lumberyard Tools, choose Animation Manager.
2. In the Animation Manager dialog box, choose the + button and then specify the following properties:
   a. For Start and End, enter values for the starting and ending frames for the animation, as defined in the Maya Range Slider settings. Choose the < > button to populate the start and end fields with the Maya time range slider start and end values.
   b. For Name, type a name for the animation.
   c. For Root Node select the root joint for animation and choose the + button.
   d. Under Animation Layers, select Selected1 from the drop-down list and then select a layer. Select or deselect BaseAnimation as applicable if the animation is primary or secondary (additive).
   e. For Export Path, choose the folder icon and select a directory path.
   f. To delete an animation from the list, choose the x button next to it.
3. Repeat Step 2 as needed for each animation you want to export.
4. Make sure the check box is selected for each animation you want to export, then choose Export Animations.

Note
To export all static geometry, materials, and animated geometry that are listed and selected in each of the three lists at once, choose Export All.

Exporting Blendshapes

The following requirements must be observed when exporting a blend shape to Lumberyard.

- Select the skinned mesh with the blend shape nodes and add it to the Geometry Export list in the Lumberyard Tools. Be sure that it has been assigned the .SKIN extension type.
- Assign the appropriate materials to your skinned blend shape meshes. It should be identical to the materials used on your main skinned mesh.
- Add a material group for the .SKIN in Materials Export if you haven’t already.
- Export the .skin file.

Exporting a Level of Detail (LOD)

Level of detail (LOD) techniques are used to increase performance and reduce draw calls by displaying progressively lower detailed objects the further they are from the camera. Generally, each LOD should have its vertices reduced 50% from the previous level and a reduction in the number of materials used. Lower LODs should also have multiple textures combined into one texture to further reduce draw calls.

Lumberyard supports using up to six LODs per group node in Maya. LOD number is from 0 (highest level of detail) to 5 (lowest level of detail).

You can export the following LODs from Maya to Lumberyard:
• Static Mesh LODs
• Material LODs
• Skinned Mesh LODs

Static Mesh LODs

Use the following naming and setup to export Static Mesh LODs using the Maya Lumberyard tool.

LOD Naming

Use the following naming conventions for LODs.

- _lod0_ through _lod5_ (prefix)
- _group_ (suffix)
- _helper_ (suffix)

LOD Setup

**Basic LOD Setup:** All LOD meshes need to be under a group node in Maya. You will need to add the _group_ suffix at the end of the name for your group node. The following example shows assets that have no animated parts and small assets that do not need to be split up for culling.
Advanced LOD Setup: Each set of LOD meshes needs to be under a group node in Maya. You will need to add the `_lod#_ prefix at the beginning and `_group` suffix at the end of the name for your group nodes that contain these sets. The following figure shows an asset that has multiple meshes and a collision mesh that must be split into LODs that can be culled. The lowest LOD does not contain a Decals_mesh because by this LOD it will not be noticeable and the removal of it will save on performance.
Note
When exporting, under the Advanced Options panel, ensure that the Merge Nodes check box is not selected.
Material LODs

When you set up material LODs, you create submaterials and assign them to the appropriate LOD mesh. For example, you might have a material group named Tree_Mats.

Within Tree_Mats, you set up five submaterials. The first three submaterials you assign to LOD0, which is the highest poly model. The other two submaterials you assign to the LOD1 mesh. This means that when LOD0 is active, it uses the first three submaterials you assigned to it. When LOD1 is active, it uses only the two submaterials you assigned to it.

Skinned Mesh LODs

Use the following naming conventions for Skinned Mesh LODs.

- LOD0: Any name (no suffix or prefix required)
- LOD1 through LOD5: Same name as LOD0 and suffixed with _lod1 through _lod5
- Optional: _group node suffix to organize meshes in Maya (not required for exporting to Lumberyard)

When you export your skinned mesh LODs, each skin exports as an individual .skin file. The following images show how a set of exported skins look in each application.

**Maya Outliner:**

![Maya Outliner](image)

**Lumberyard Exporter in Maya:**

**Windows Explorer:**
Exporting a Level of Detail (LOD)

You do not need to set anything else for the LODs within the .CDF. This is because when you assign the LOD0 (highest poly) skin mesh, the engine automatically handles swapping LOD meshes that follow the same naming convention for skins in the same folder.

To export skinned mesh LODs

1. Ensure that your LOD0 through LOD6 meshes are named and suffixed appropriately. See the naming conventions described in the introduction.
2. Open the Maya Lumberyard Tools. Make sure the Export window is visible.
3. Select your LOD0 mesh. Click Add Selected to add it to the Geometry Export list.
   
   If it has skin weight data, the export type is automatically set to .SKIN.
4. Select your LOD1 mesh. Click Add Selected to add it to the Geometry Export list.
   
   If it has skin weight data, the export type is automatically set to .SKIN.
5. Repeat for the remaining LOD meshes.
6. After you add all your LOD skinned meshes, assign the appropriate export path. Then export your skeleton (.CHR) as you would normally.

<table>
<thead>
<tr>
<th>Name</th>
<th>Date modified</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>body_mesh.skin</td>
<td>6/14/2017 1:41 PM</td>
<td>ASP.NET Skin File</td>
<td>624 KB</td>
</tr>
<tr>
<td>body_mesh_loc1.skin</td>
<td>6/14/2017 1:41 PM</td>
<td>ASP.NET Skin File</td>
<td>451 KB</td>
</tr>
<tr>
<td>body_mesh_loc2.skin</td>
<td>6/14/2017 1:41 PM</td>
<td>ASP.NET Skin File</td>
<td>351 KB</td>
</tr>
<tr>
<td>dummyPlane_mat_group.mtl</td>
<td>6/13/2017 4:54 PM</td>
<td>MTL File</td>
<td>1 KB</td>
</tr>
<tr>
<td>guy_group_mats.mtl</td>
<td>6/13/2017 4:44 PM</td>
<td>MTL File</td>
<td>1 KB</td>
</tr>
<tr>
<td>head_mesh.skin</td>
<td>6/14/2017 1:41 PM</td>
<td>ASP.NET Skin File</td>
<td>72 KB</td>
</tr>
<tr>
<td>head_mesh_loc1.skin</td>
<td>6/14/2017 1:41 PM</td>
<td>ASP.NET Skin File</td>
<td>66 KB</td>
</tr>
<tr>
<td>head_mesh_loc2.skin</td>
<td>6/14/2017 1:41 PM</td>
<td>ASP.NET Skin File</td>
<td>42 KB</td>
</tr>
<tr>
<td>new_skel chr</td>
<td>6/13/2017 4:54 PM</td>
<td>3dsMax character...</td>
<td>29 KB</td>
</tr>
<tr>
<td>skin_test.cdf</td>
<td>6/14/2017 1:58 PM</td>
<td>CDF File</td>
<td>1 KB</td>
</tr>
</tbody>
</table>
Exporting a Level of Detail (LOD)

Note
All the LOD skinned meshes must be in the same directory for the LODs to work properly in Lumberyard.

7. Open Lumberyard. Set up your `.CDF`, `.CHR` and `.SKIN` attachment(s) as normal.

If the proper naming convention was followed, Lumberyard automatically handles the transition of the LOD meshes in-level.

To test the `.CDF` with LODs, drag the `.CDF` into the level and then move your camera back to watch the LODs transition.

Currently, there is no way to test the LODs within Geppetto. You can use the console variable `e_DebugDraw = 1` to display debug text information regarding LODs. However, it may difficult to read if you have multiple skin attachments that have LODs, as the text stacks on top of each other.
LOD Distance Ratio

The LOD Distance Ratio is a ratio derived from the Max view distance and View distance multiplier values.

To easily test the LOD distance ratio's relation to the maximum view distance, leave the distance multiplier set to 1. A higher LOD distance ratio results in faster transitions through the LODs as the camera pulls away. Lower LOD distance ratio values effect slower transitions through the LODs.

For example, LOD transitions was tested for a tree that has LOD0-LOD2 and a Max view distance of 100 (m) and LOD distance ratio of 255. The tree transitioned from LOD0 to LOD1 around 15m away, and then from LOD1 to LOD2 around 25m away. When LOD distance ratio was set to 64, the tree transitioned from LOD0 to LOD1 around 40m away, and then from LOD1 to LOD2 around 75m away. Experimenting with these values showed that the calculation is not linear.

Debugging LODs

The following console variables can be used for debugging LODs:

- e_DebugDraw = 1 - Name of the used cgf, polycount, and the used LOD.
- e_LodCompMaxSize = 6 - (default value) Lower values force the LODs to load sooner.
- e_LodRatio = 40 - (default value) LOD distance ratio for objects. Lower values force LODs to load sooner.
- r_Stats = 6 - Displays the drawcall count for each object instance in the scene. The numbers above each object are broken into total DP, zpass, general, transparent, shadows, and misc.

Exporting an Alembic Cache

Alembic distills complex, animated scenes into a non-procedural, application-independent set of baked geometric results. Specifically, it handles baked meshes and their materials, but not rigs or other procedural networks of computations.

Lumberyard allows you to export Alembic (.abc) cache files from Maya. Lumberyard then compiles them into compressed game asset (.cax) files using the Resource Compiler and imports them into the game using the GeomCache entity. In-game, the .cax files are then streamed off disk.

Note
Deforming meshes can be exported along with their UVs and material assignments. However, multiple UV sets are not exported; only the default UV set is exported.

To export an Alembic cache from Maya

1. In Maya, rename each material using a unique integer ID. Material names are scanned from left to right and the first integer found is used. For example: mat01_sphere, mat02_sphere, mat03_cube.
3. In Lumberyard Tools, under Material Export, choose Add Group and then enter a name. The name of this material group (.mtl) file must match the name of the exported Alembic (.abc) cache file.
4. Set the export path to any folder within your game directory, and then choose Export Materials.
5. In Maya, select the geometry objects you want to export, and then in Lumberyard Tools, choose Tools, Export Select to Alembic.
6. In Export Alembic for Geomcache, navigate to the same directory used to export the materials to, enter the same name used for the material group, and then choose Save.
Lumberyard imports Alembic caches using the **GeomCache** entity found in the Rollup Bar.

**To import an Alembic cache to Lumberyard**

1. In Lumberyard Editor, choose New and then enter a name for the new level.
2. In the Rollup Bar, on the Objects tab, choose Entity.
3. Under Browser, expand Render. Select GeomCache, drag it into the level, then click to position the entity.
4. Under Entity Properties, choose the folder icon for File, select the Alembic (.abc) cache file previously exported, and then choose Open.
5. In Compile Alembic, change preset, compilation, and compression settings as needed, and then choose OK.
6. In Running Resource Compiler, review and resolve any errors, and then choose Close.

**Note**

Lumberyard automatically changes the File property to point to the compiled .cax file. If you modify the Alembic (.abc) cache file later, you'll need to recompile it into a .cax file. To do this from Lumberyard Editor, change the File property to point to the .abc file instead of the .cax file. You will then be prompted to repeat the steps in this section.

**Setting Export Options**

Lumberyard has a number of options to customize the export process. To apply them, select a geometry node from the list, choose Tools, and select from the following as needed.

**Add Scene Root**

Creates a scene node that re-orients exported nodes relative to the displayed orientation.

**Move Origin to Pivot**

Sets a selected object's transform as an offset from the origin. If the Center Pivots check box is enabled, it will also center the pivot of the selected object.

**Zero Joint Rotations**

Removes any rotations on the selected joint and sets the value to zero.

**Add Attributes**

Exposes Lumberyard variables to joints and materials.

**User Defined Properties**

Opens a dialog box to add custom properties that is most commonly used for assigning a defined collision shape (sphere, box, or capsule) to override the existing collision mesh shape.

**Polygon Check**

Checks for degenerate faces.

**Prepare Alembic Materials**

Slightly modifies a scene to work around limitations in the Maya Alembic Exporter by changing the scene's shading engines and shading groups to enable the export of faceset information, which is used for the transport of the material assignments.

**Export Selected to Alembic**

Exports geometry caches that allow storing and playing arbitrarily animated geometry.
Joint Proxy Editor (Experimental)

Opens the Lumberyard Proxy Tool, which is used to create physics proxies for characters to be physicalized.

Validator

Runs the validation process.

3ds Max Export Tools

Lumberyard has a plugin for Autodesk 3ds Max 2014–2016 to simplify exporting static geometry, character geometry, and materials to Lumberyard. To install this plugin, go to the Lumberyard root directory (\lumberyard\dev) and start Lumberyard Setup Assistant, choose Integrated tools, and then choose Autodesk Max.

Topics

- Exporting Static Meshes and Characters (p. 26)
- Exporting Materials (p. 28)
- Exporting Bones and Animations (p. 28)
- Exporting a Level of Detail (LOD) (p. 31)
- Configuring Log Options (p. 32)

Exporting Static Meshes and Characters

Use the following procedure to export geometry and character geometry. You can specify which nodes in the scene to export, and other options about how they are exported. Child nodes are also exported.

When exporting character skin meshes, the 3ds Max exporter does not support combining multi-skin meshes into a single .skin file. You should export your multi-skin meshes as separate .skin files. To do so, add your skin meshes to the Geometry Export list and select the Export file per node check box. This will export each node in the list as a separate .skin file.

Note

If you are exporting multiple Proxy No Draw meshes, they must be children of a single object, such as a dummy object. This ensures that the exported meshes have collision functionality in Lumberyard.
To set geometry export options for 3ds Max

1. In 3ds Max, click the Utilities tab (hammer icon), and then choose More.
2. In Utilities, double-click Lumberyard Exporter.
3. In Geometry Export, choose the node in the viewport, and then choose Add Selected. Repeat as needed.
4. Choose the desired options as listed in the following table and then choose Export Nodes.

**Geometry Export Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Export Format</td>
<td>Specify the file format for the exported file. Geometry export file formats include geometry (<em>.cgf), character (</em>.chr), character skeleton (<em>.skel), and character skin (</em>.skin).</td>
</tr>
<tr>
<td>Export file per node</td>
<td>Export each node in the export list as a separate file. The filename is generated from the node name.</td>
</tr>
</tbody>
</table>
Exporting Materials

Use the following procedure to export materials.

**Note**
When exporting a multi-material with one of the sub-materials for your proxy collision, you must assign the proxy collision sub-material to **MaterialID 1**. This allows the proxy collision mesh to process correctly in Lumberyard.

To export materials

1. In 3ds Max, choose the **Utilities** tab (hammer icon), and then choose **More**.
2. In **Utilities**, double-click **Lumberyard Exporter**.
3. In **Materials**, do one of the following:
   - To update 3ds Max material settings to match those used in the Lumberyard material .mtl file for the object, choose **Sync Material**.
   - To create a material .mtl file with settings that match those used for the 3ds Max material, choose **Create Material**.

Exporting Bones and Animations

**Animation Export** contains the settings for exporting skeleton and animations for skinned character models. When you add a node to the **Geometry Export** list, its skeleton root bone is also added to the **Animation Export** list. When exporting your skeleton bones (.chr), you should have only one node in the **Geometry Export** list.
You typically don't need to configure the **Animation Export** settings if you have your skin mesh node listed in the **Geometry Export** list. However, the ability to directly edit this list may be helpful. For example, if you want to export animations for only the upper body.

**Note**
You must export your animations using the **30 FPS** frame rate setting, otherwise the Asset Processor will fail. You can set this value in the **Time Configuration** dialog under **Frame Rate**.

Use the following procedure to export character skeleton bones. Child bones are also exported.

![Bone Export Options](image)

**To set bone export options for 3ds Max**

1. In 3ds Max, choose **Utilities** tab (hammer icon), and then choose **More**.
2. In **Utilities**, double-click **Lumberyard Exporter**.
3. In **Geometry Export**, choose the node in the viewport, and then choose **Add Selected**.
4. In **Animation Export**, choose the desired options as listed in the following table, and then choose **Export Bones**.

<table>
<thead>
<tr>
<th>Bone Export Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>use object bones (Lock)</td>
<td>Use the bone of the geometry target listed in <strong>Geometry Export</strong>.</td>
</tr>
<tr>
<td>Ignore Dummy bones</td>
<td>Prevent exporting any dummy bones that are in the bone hierarchy.</td>
</tr>
</tbody>
</table>

5. (Optional) In order to export animation data, within the **Animation range** you can specify the animation range for a character's skeleton. Use the various parameter options in the following table.
### Animation Range parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entire timeline</td>
<td>Use the full timeline length.</td>
</tr>
<tr>
<td>Custom</td>
<td>Use the customized length by specifying the start and end frames.</td>
</tr>
<tr>
<td>Multiple custom ranges</td>
<td>Use specified multiple animation ranges (for details, see the following procedure).</td>
</tr>
</tbody>
</table>

#### To edit multiple custom animation ranges

1. In **Animation Export**, choose **Edit custom ranges**.
2. In **Animation Sub-Ranges**, double-click **<New Range>** and then type a name.
3. Use the arrows to specify the start and end frames.
4. Click the (...) icon and then choose an export file path for the animation range.
Exporting a Level of Detail (LOD)

Level of detail (LOD) is a technique that increases performance and reduces draw calls by displaying progressively lower detailed objects the further they are from the camera. Generally, each LOD should have its vertices reduced 50% from the previous level and a reduction in the number of materials used. Lower LODs should also have multiple textures combined into one texture to further reduce draw calls.

Lumberyard supports up to six LODs per group node in 3ds Max. The LOD numbering is from 0 (highest level of detail) to 5 (lowest level of detail).

You can export the following LODs from 3ds Max to Lumberyard:

- Static Mesh LODs
- Material LODs

Static Mesh LODs

Use the following naming and setup to export Static Mesh LODs using the 3ds Max Lumberyard tool.

LOD Naming

LOD naming conventions are very important with respect to prefixes and suffixes. You must use the following naming conventions:

The highest LOD mesh (LOD 0) does not need a prefix.

$lod1_ through $lod5_ (prefix)

LOD Setup

You can create a basic or advanced LOD setup.

Basic LOD Setup – All LOD meshes with the appropriate prefix must be parented under the main render mesh (LOD0). Refer to the example below for assets that have no animated parts or for small assets that do not need to be split up for culling.

Advanced LOD Setup – When you have LOD subobject meshes, the same rule applies as the basic setup where the LOD meshes with the appropriate prefix must be parented under their respective main render mesh (LOD0). The LOD0 mesh for the subobjects should be parented under the main object LOD0 mesh. Refer to the example below for assets that have animated parts or that are large and need to be split into multiple objects that can be culled.
Material LODs

When you set up material LODs, you create submaterials and assign them to the appropriate LOD mesh.

For example, you might have a material group named `Tree_Mats`.

Within `Tree_Mats`, you set up five submaterials. The first three submaterials you assign to LOD0, which is the highest poly model. The other two submaterials you assign to the LOD1 mesh. This means that when LOD0 is active, it uses the first three submaterials you assigned to it. When LOD1 is active, it uses only the two submaterials you assigned to it.

Debugging LODs

Use the following console variables to debug LODs:

- `e_DebugDraw = 1` – Name of the used cgf, polycount, and the used LOD.
- `e_LodCompMaxSize = 6` – (default value) Lower values force the LODs to load sooner.
- `e_LodRatio = 40` – (default value) LOD distance ratio for objects. Lower values force LODs to load sooner.
- `r_Stats = 6` – Displays the drawcall count for each object instance in the scene. The numbers above each object are broken down into total DP, zpass, general, transparent, shadows, misc.

Configuring Log Options

There are several options for configuring what is logged during export.

To set exporter log options for 3ds Max

1. In 3ds Max, choose the Utilities tab (hammer icon), and then choose More.
2. In Utilities, double-click Lumberyard Exporter.
3. In **Geometry Export**, choose the node in the viewport, and then choose **Add Selected**. Repeat as needed.

4. In **Options**, choose the desired options as listed in the following table, and then choose **Show Log**.

![Options](image)

### Other Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degenerate UVW</td>
<td>Check for degenerate texture coordinates and issue a warning if they exist; otherwise, silently export them. Degenerate coordinates arise when two vertices on a triangle have the same (or very nearly the same) UVs.</td>
</tr>
<tr>
<td>Off-axis scaling</td>
<td>Check whether the node is scaled along a non-primary axis. You can still export the node, but the scale won’t match the object in 3ds Max.</td>
</tr>
</tbody>
</table>

**Using Geppetto**

Geppetto is used to set up fully animated characters in Lumberyard, in preparation for use with either custom game code to select, play, and blend animations, or with the Mannequin animation controller system. In Geppetto you build a character by associating one or more skinned models with an animation skeleton (built in a DCC like 3DS Max or Maya), and specifying a list of animations (built in a DCC like 3DS Max or Maya) to use with that character.

Animations can also be combined together into blend spaces, which are collections of similar animations that can smoothly blend together to vary travel speed, turning speed, travel angle, slope, turn angle, and travel distance. Blend spaces allow you to easily author natural, complex locomotion for characters. You can use Geppetto to add attachments to the character, such as weapons or other props, including physically simulated attachments that are connected by springs, pendulums, and strings of joints, allowing you to model clothing attachments, capes, and large scale movement of hair. Geppetto also allows you to preview animations and blends between animations on the characters you define, set compression settings for game ready animation data, and compare compressed and uncompressed animations.
To access Geppetto from Lumberyard Editor, choose **Tools, Geppetto**. Geppetto has the following UI:
Lumberyard Legacy Reference
Using Geppetto

A. Viewport window
Displays the loaded character. Use the WASD keyboard keys for movement and the right mouse button for camera rotation.

B. Assets pane
Lists all character assets, skeletons, animations, and compression settings. Each asset item has a context menu with available options. When an asset is selected, its properties are displayed in the Properties panel.

There are multiple ways to filter the tree in the Assets panel:

- By name. It is possible to specify multiple strings separated by a space to look for substrings. For example, walk relaxed looks for any name that contains both "walk" and "relaxed".
- By type
- Using advanced filtering options, like presence of events or location of file.

You can have multiple instances of the Assets window open. To create a new instance, choose Split Pane Assets navigation bar.

C. Scene Parameters panel
This panel is used for previewing purposes and consists of the following:
• **Character name** – Used to select and load a new character by clicking the folder icon. When a character is loaded, you can use the button to select a CDF so you don't have to locate it in the Assets tree every time.

• **Animation Layers** – Location where the played animations are set up. Whenever you select an animation in the Assets panel, one is assigned to the active animation layer, which is highlighted with bold text. Add new animation layers using the button next to it. Remove animation layers through the context menu. Blend spaces, aimposes, and lookposes expose additional settings.

• **Blend shape** – Shows blend shape sliders when the character contains blend shapes.

• **Audio Setup** – Used to preview sound foleys and footsteps.

• **Run Feature Test** – Used to add and run project-specific tests.

**D. Properties panel**

Lists character definitions, skeleton, and animation properties.

**E. Playback panel**

Displays the animation timeline and playback options, such as looping and speed.

All panels can be moved and are dockable.

**Note**

Because hot reloading of character-related assets is not supported in Lumberyard Editor, you need to close and restart Lumberyard Editor if you modify any characters that pre-exist in a level. This is not necessary for characters that later spawn into a level. This does not apply if you are previewing changes in Geppetto.

**Topics**

- Geppetto Display Options (p. 36)
- Creating a Character Definition (p. 40)
- Character Attachments (p. 41)
- Animating Characters (p. 60)

**Geppetto Display Options**

The following is a list of display option settings in Geppetto. In Geppetto, choose **Display Options** to access the various settings.

**Animation**

You can specify the following animation options:

**Movement**

Choose between **In Place (Only grid moves)**, **Repeated**, and **Continuous (Animation Driven)** in response to when the character’s root joint moves in world space during an animation.

**Compression**

Choose between **Preview Compressed Only** and **Side by Side (Original and Compressed)** for what to preview for animations.

**Animation Event Gizmos**

Enables and disables the visibility of animation event gizmos that are tied to a skeleton joint.
**Locomotion Locator**

Enables and disables the visibility of the locomotion locator for the character, to indicate which direction the root motion or locomotion locator are pointing during an animation.

**DCC Tool Origin**

Enables and disables the transform display on the DCC origin for the skeleton and also displays the rotation and position information near the top of the viewport.

**Reset Character**

Allows you to reset the character by forcing it back to bind pose, setting it to viewport origin, and removing any current animations on the character, including the removal of animation layers.

**Rendering**

You can specify the following rendering options:

**Edges**

Enables and disables the display of all edges for polygons on meshes. It also displays information regarding the mesh data at the top of the viewport.

**Wireframe**

Enables and disables the wireframe mode for meshes. If used in combination with **Edges**, it uses a flat colored wireframe instead of the material wireframe.

**Framerate**

Enables and disables the display of the frame rate for the viewport.

**Skeleton**

You can specify the following skeleton options:

**Joint Filter**

Allows you to enter text to help filter what joints are displayed so joints are only displayed that have the text somewhere in the joint name. Should be used with **Joints** enabled.

**Joints**

Enables and disables the display of skeleton joints.

**Joint Names**

Enables and disables the display of skeleton joint names.

**Bounding Box**

Enables and disables the display of the bounding box for the character created by the skeleton joints.

**Camera**

You can specify the following camera options:
**Show Viewport Orientation**

Enables and disables the display of the viewport orientation.

**FOV**

Slider to adjust the camera's FOV.

**Near Clip**

Slider to adjust the camera's near clip plane.

**Move Speed**

Slider to adjust the movement speed of the camera, currently capped at 3. The default is 0, not restraining the camera at all. If this parameter is set to an odd number, it does not allow the use of rotation for the camera.

**Rotation Speed**

Slider to adjust the rotation speed of the camera.

**Movement Smoothing section**

You can specify the following movement smoothing options:

**Position**

- Slider for adjusting smoothing for the camera translation.

**Rotation**

Slider for adjusting smoothing for the camera rotation.

**Follow Joint**

You can specify the following follow joint options:

**Joint**

Joint that the camera will follow. The default is null so that you can manipulate the camera.

**Align**

Enables and disables the alignment of the camera to the specified joint based on Position and Orientation.

**Position**

Enables and disables the position of the joint to influence the camera.

**Orientation**

Enables and disables the orientation of the joint to influence the camera.

**Secondary Animation**

You can specify the following secondary animation options:

**Dynamic Proxies**

Enables and disables the display of dynamic proxies.
**Auxiliary Proxies**

Enables and disables the display of auxiliary proxies.

**Physics**

You can specify the following physics options:

**Physical Proxies**

Enables and disables the display of physics proxies.

**Ragdoll Joint Limits**

Enables and disables the display of the ragdoll joint limits on the skeleton.

**Grid**

You can specify the following grid options:

**Show Grid**

Enables and disables the display of the grid. There are additional settings for setting the grid main line and middle line color and transparency.

**Spacing**

Sets the scale of the grid based on meters. The default is 1.

**Main Lines**

Sets the display of the number of grid main sections.

**Middle Lines**

Sets the display of the number of middle sections within the grid main sections.

**Origin**

Enables and disables the display of the viewport origin. When enabled, this parameter gives additional options for adjusting the color and transparency of the origin.

**Lighting**

You can specify the following lighting options:

**Brightness**

Sets the brightness of the light. You also have control over the color and transparency of the light.

**Rotate Light**

Enables and disables the rotation of the light in world space.

**Light Multiplier**

Sets the multiplier for the light.

**Light Spec Multiplier**

Sets the multiplier for the specular for the light. You also have control over the color and transparency of the specular for the light.
Background

You can specify the following background options:

**Use Gradient**

Enables and disables the use of gradient with the colors assigned below. If disabled, only one color is available to adjust.

Creating a Character Definition

Using Geppetto, you can create a character definition. The character definition .cdf file consists of a skeleton .chr file, an animation list that is referenced in a .chrparams file, and attachments.

Character Definition File

The XML-based character definition file (.cdf) combines different character parts such as skeletons, meshes, materials, and attachments.

Before proceeding, make sure you have the following assets exported from your DCC tool:

- Character skeleton .chr file
- Skinned geometry .skin file
- One or more character animations

**To create a character definition file**

1. In Geppetto, choose File, New Character, type a file name and path, then choose Save. An empty file is created, but without a skeleton or attachment yet.
2. In the Properties panel, choose the folder icon next to Skeleton, select the skeleton .chr file, and choose Open to load the skeleton. This assigns the skeleton to the .cdf file.
3. Choose Display Options to reveal the Skeleton section in the UI.
4. Expand Skeleton and choose Joints. The skeleton is displayed in the viewport.

Character Skeleton List

Make sure the skeleton is added to the SkeletonList.xml file using the following procedure.

**To add the skeleton to the list**

1. In the Assets panel under Compression (Animations), choose Skeleton List.
2. In the Properties panel under Aliases, make sure the skeleton .chr file is in the list. If not, do the following:
   a. Choose the number button next to Aliases and Add.
   b. Choose the folder icon next to the new entry, then select a suitable skeleton.
   c. Name the added skeleton alias. This name is used to refer to the skeleton.

Character Animation List

The character animation list is specified in the .chrparams file.
To specify the animation list
1. In the Asset panel, expand Skeletons, Characters and select the skeleton .chr file.
2. In the Properties panel, choose the number button next to Animation Set Filter and Add.
3. Select the folder icon for the new row, open the context (double-click) menu for Animations, and then choose Select Folder.

Character Attachments

In order to attach something to a character, a socket is needed. Sockets provide the connection between the character and the attachment. For more information, see Attachment Sockets (p. 41).

After a socket has been created and defined, an attachment can be created and connected to the socket. For more information, see Character Attachments (p. 41).

Character Attachments

Attachments are separate objects that are attached to characters, respond to real-world physics, and can be attached, detached, or replaced at runtime in the game.

Lumberyard allows for various skinned, animated, or physicalized attachments to the skeleton or polygonal faces of a character. Attachments are hierarchical, and each attachment can have its own morph targets and animations. You can use skin attachments for entire body parts such as heads, hands, or upper and lower body.

To add or change a character attachment, the character must first be loaded into Geppetto.

Topics
- Attachment Sockets (p. 41)
- Joint Attachments (p. 42)
- Face Attachments (p. 43)
- Pendula Row (PRow) Attachments (p. 43)
- Proxy (Collision) Attachments (p. 45)
- Skin Attachments (p. 46)
- Collision Detection and Response (p. 50)
- Secondary Animations (Simulations) (p. 55)

Attachment Sockets

To attach something to a character, you must first create an attachment socket. A socket is an empty attachment without assigned geometry. Sockets have a name, position/orientation (for joint and face attachments), and attachment type. Attachment sockets can be used by game code to attach objects to characters at runtime, such as replacing weapons attached to a hand. After a socket is created, you can plug a .cgf attachment into it.

Tip
To display all empty sockets for a character, use the ca_DrawEmptyAttachments=1 console variable.

You can also use sockets to achieve simulated motion of joint and face attachments. This type of animation is always a reaction to a primary character motion, and are called secondary animations.
Such animations can simulate the movement of attached objects. For more information, see Secondary Animations (Simulations) (p. 55).

**Joint Attachments**

Joint attachments require an attachment socket that provides a connection point between the attachment and the character. Use the move and rotate tool to position and orient the socket relative to a bone joint.

The socket is attached to a joint and moves with the joint when the skeleton is animated. Secondary animations can be enabled on a socket and provide additional motions based on a real-world physical simulation and generated in response to the movements of the character. This has the effect of making loosely-attached objects behave more realistically when the character is undertaking fast movements.

These secondary animations can also be redirected to the skeleton of the character to apply the simulated motion to all vertices that are part of the skinned mesh and weighted to the joint. This is very useful when animating hair and cloth. By enabling collision detection, such attachments can also interact with the character.

You can simulate the motion of hair braids and dangling straps using joint attachments. A chain or rope of pendula can be created by attaching a pendulum at each link. When the motion simulation is activated, each parent joint transfers motion to the children. In this case, the primary motion is not coming from an animation, but from a previous motion simulation. Collision detection and response is used to limit the motion of the attachment from moving through the body of the character.

**To create a joint attachment**

1. In Geppetto, in the Properties panel, click the number next to Attachments and Add or Insert.
2. For Name, enter a name for the attachment.
3. For Type, choose Joint Attachment.
4. For Joint, choose the bone icon, then open the applicable joint to place the socket on.
5. For Geometry, choose the folder icon and select the desired *.cgf file for the attachment.
6. For Material, choose the folder icon and select the desired *.mtl file for the attachment.
7. Adjust the values of attachment parameters for the desired result, as listed in the following table.

**Joint Attachment Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>View Distance Multiplier</td>
<td>Multiplier to the computed fade-out camera distance to apply on the attachment.</td>
</tr>
<tr>
<td>Store Position</td>
<td>Stores position data relative to either the character or to a joint.</td>
</tr>
<tr>
<td>Store Rotation</td>
<td>Stores rotation data relative to either the character or to a joint.</td>
</tr>
<tr>
<td>Transform</td>
<td>Transform-Translation (T) and Rotation (R) vectors for the X, Y, and Z axes in relation to Store Position and Rotation.</td>
</tr>
<tr>
<td>Simulation</td>
<td>Type of simulated motion. Disabled is on by default, but types consist of: Pendulum Cone, Pendulum Hinge, Pendulum Half Cone, Spring Ellipsoid, and Translational Projection. For more information, see Secondary Animations (Simulations) (p. 55).</td>
</tr>
<tr>
<td>Hidden</td>
<td>Hides the attachment.</td>
</tr>
<tr>
<td>Physicalized Rays</td>
<td>Enables hit ray detection if a physics proxy is available.</td>
</tr>
</tbody>
</table>
Character Attachments

Parameter | Description
---|---
Physicalized Collisions | Enables collision detection if a physics proxy is available.

Face Attachments

Face attachments require an attachment socket that provides a connection point between the attachment and the character. The socket is attached to a specific triangle on the mesh surface and moves along with the triangle when the skeleton is animated and the mesh gets deformed. The location of the face attachment can be relative to the triangle and it is possible to assign face attachments to all skinned meshes of a character.

It is recommended that the character be first put into its bind pose. To do so, in Geppetto, in the Scene Parameters panel, choose Bind Pose next to Animation Layers.

When you move the socket using the using the gizmo tool in the viewport, it automatically connects to the closest triangle in the mesh.

To create a face attachment

1. In Geppetto, in the Properties panel, click the number next to Attachments and Add or Insert.
2. For Name, enter a name for the attachment.
3. For Type, choose Face Attachment.
4. For Geometry, choose the folder icon and select the desired *.cgf file for the attachment.
5. For Material, choose the folder icon and select the desired *.mtl file for the attachment.
6. Adjust the values of attachment parameters for the desired result, as listed in the following table.

Face Attachment Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>View Distance Multiplier</td>
<td>Multiplier to the computed fade-out camera distance to apply on the attachment.</td>
</tr>
<tr>
<td>Transform</td>
<td>Transform-Translation (T) and Rotation (R) vectors for the X, Y, and Z axes in relation to Store Position and Rotation.</td>
</tr>
<tr>
<td>Simulation</td>
<td>Type of simulated motion. Disabled is on by default, but types consist of: Pendulum Cone, Pendulum Hinge, Pendulum Half Cone, Spring Ellipsoid, and Translational Projection. For more information, see Secondary Animations (Simulations) (p. 55).</td>
</tr>
<tr>
<td>Hidden</td>
<td>Hides the attachment.</td>
</tr>
<tr>
<td>Physicalized Rays</td>
<td>Enables hit ray detection if a physics proxy is available.</td>
</tr>
<tr>
<td>Physicalized Collisions</td>
<td>Enables collision detection if a physics proxy is available.</td>
</tr>
</tbody>
</table>

Pendula Row (PRow) Attachments

To create a pendula row attachment

1. In Geppetto, in the Properties panel, choose the number next to Attachments and Add or Insert.
2. For **Name**, enter a name for the attachment.
3. For **Type**, choose **PRow Attachment**.
4. For **Joint Row Name**, choose the bone icon, then open the applicable joint to place the socket on.
5. Adjust the values of attachment parameters for the desired result, as listed in the following table.

**Pendula Row Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clamp Mode</td>
<td>Used to select the movement bounding volume of the pendula row: Cone, Half Cone, Hing, or Translational Projection.</td>
</tr>
<tr>
<td>Debug Setup</td>
<td>When enabled, displays a green bisected spherical shape that represents the bounding volume for the simulated object's pivot.</td>
</tr>
<tr>
<td>Debug Text</td>
<td>Enable to display debugging text in the viewport.</td>
</tr>
<tr>
<td>Activate Simulation</td>
<td>Used to activate the physics simulation for springs and pendula.</td>
</tr>
<tr>
<td>Simulation FPS</td>
<td>Used to specify the frame rate of the physics simulation updates. A value of 30 indicates 30 updates per second. The valid value range is 10–255 fps, with a recommended range of 30–60 fps. This value should ideally be the same as the game frame rate.</td>
</tr>
<tr>
<td>Mass</td>
<td>Used to specify the mass of pendula bobs. If the value of the Joint Spring parameter is zero, the Mass value has no impact on the oscillation period.</td>
</tr>
<tr>
<td>Gravity</td>
<td>Used specify the force of gravity on pendula. While the mass of a bob has no effect on the oscillation of a pendulum, the force of gravity does. The default value of 9.81 represents Earth's gravitational force.</td>
</tr>
<tr>
<td>Damping</td>
<td>Used to specify a velocity-dependent force such as air resistance. The faster that pendula move, the more force that is encountered, decelerating the pendula at a rate proportional to the velocity. Greater damping values result in pendula coming to rest more quickly.</td>
</tr>
<tr>
<td>Joint Spring</td>
<td>Used to simulate position dependent forces, and is a value between 0–999 applied to the spherical joint. The further the pendulum swings away from the axis of the spring target, then the harder it tries to return.</td>
</tr>
<tr>
<td>Cone Angle</td>
<td>Used to specify the pendula starting movement angle for cone, half-cone, and hinge-planes bounding volumes. Valid range is from 0–179 degrees, where values greater than 90 degrees form an inverse cone.</td>
</tr>
<tr>
<td>Cone Rotation</td>
<td>Used to specify the amount of rotation relative to joints along the X, Y, and Z axes for cone, half-cone, and hinge-planes.</td>
</tr>
<tr>
<td>Rod Length</td>
<td>Used to specify the length of pendula row rods, which impacts swinging frequency. The longer the rods, the longer the pendula oscillations.</td>
</tr>
<tr>
<td>Spring Target</td>
<td>Used to specify two planes of rotation around the x-axis of the joints.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Turbulence</td>
<td>Used to control frequency and amplitude of noise added to PRow joints to simulate wind and similar effects on cloth.</td>
</tr>
<tr>
<td>Max Velocity</td>
<td>Used to clamp the velocity of the PRow pendula bobs in order to control large impulse spikes from character movements.</td>
</tr>
<tr>
<td>Cycle</td>
<td>Select to attach the last joint in the pendula row to the first joint to form a horizontal circle. Used for cloth skirts.</td>
</tr>
<tr>
<td>Stretch</td>
<td>Used to define the horizontal distance between pendula row joints, which defines how much cloth can stretch or shrink horizontally. A value of 0.2 indicates a stretching or shrinking of 20%.</td>
</tr>
<tr>
<td>Relax Loops</td>
<td>Used to iteratively keep pendula row joints together horizontally. Each iteration brings the joints closer together for each frame. A value between 2–4 is recommended.</td>
</tr>
<tr>
<td>Capsule</td>
<td>Defines the length and radius values for the capsules used for the dynamic (blue) proxies connected to each joint in the entire pendula row. Used for collision detection.</td>
</tr>
<tr>
<td>Projection Type</td>
<td>Choose <strong>Shortarc Rotation</strong> to activate collision detection.</td>
</tr>
</tbody>
</table>

### Proxy (Collision) Attachments

Collision detection and response involves the realistic animation of attachments that collide with the body of a living character to simulate real-world physics. To handle this, a special attachment called a collision proxy is used. Collision proxies are normal attachments that are linked to joints and move with the skeleton. Using a collision proxy is more efficient than undertaking all the necessary computation required for collision detection and response with a polygonal mesh.

Two different types of collision proxies are used:

- **Auxiliary proxies (lozenges)** – Called lozenges, these are represented in gray by simple geometric objects linked to joints that move with the skeleton, and represent an approximation of a body shape. Gray proxies handle collision detection and response with the character and are normal attachments.

- **Dynamic proxies** – These are represented in blue by capsules and spheres and are a property of a socket. Blue proxies handle collision detection and response between gray proxies. Blue proxies are dynamic collision proxies, which means that gray proxies always push blue proxies away.

Collision detection is detecting when an overlap occurs between an auxiliary proxy and a dynamic proxy. For both proxy types, you can tweak the size, shape, and other physical parameters interactively while a character animation is running and see the effect immediately.

### Auxiliary Proxies (Lozenges)

An auxiliary proxy lozenge is defined by a radius and scaling values for the X, Y, and Z axes. Using these four numbers, points, line-segments, rectangles, boxes, spheres, 1D lozenges (capsules), 2D lozenges, and 3D lozenges can be created. These eight shapes are used to approximate the shape of arms, legs, and torso of a living character.

The following figure shows a capsule shape defined for the thigh joint on the right leg of a character.
To set up an auxiliary proxy (lozenge)

1. In Geppetto, choose Display Options to reveal the Secondary Animations section, then select the Auxiliary Proxies check box.
2. In the Properties panel, choose the number next to Attachments and then choose Add or Insert.
   - For **Type**, choose Proxy Attachment.
   - For **Joint**, choose the bone icon; in the Choose Joint window, select the joint to attach the lozenge to and choose OK.
   - For **Purpose**, choose Auxiliary.
   - For **Radius**, enter a value in meters.
   - For **X-axis**, enter a value in meters.
   - For **Y-axis**, enter a value in meters.
   - For **Z-axis**, enter a value in meters.

The axes scale in both directions, so entering values of 0,1,1,1 creates a box of 2x2x2 meters.

**Dynamic Proxies**

Dynamic (blue) proxies handle collision detection and response between gray proxies. Blue proxies are dynamic collision proxies, which means that gray proxies always push blue proxies away. For more information, see Collision Detection and Response (p. 50).

**Skin Attachments**

Skin attachments have their own skeleton, so that you can replace body parts, such as heads, hands, or upper and lower body parts. Also, these body parts are automatically animated and deformed by the base skeleton. Skeleton extensions support skinned attachments that have more joints and different joints than the base skeleton. You can also merge together different types of skeletons, even skeletons from different characters.
To create a skin attachment in Geppetto

1. In Geppetto, in the Properties pane, click the number next to Attachments and then choose Add or Insert.
2. For Name, type a name for the attachment.
3. For Type, choose Skin Attachment.
4. For Geometry, click the folder icon and then select the .skin file for the attachment.
5. For Material, click the folder icon and then select the .mtl file for the attachment.
6. Specify the following skin attachment parameters for the desired result.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>View Distance Multiplier</td>
<td>Multiplier to the computed fade-out camera distance to apply on the attachment.</td>
</tr>
<tr>
<td>Hidden</td>
<td>Hides the attachment.</td>
</tr>
<tr>
<td>Software Skinning</td>
<td>If selected, the mesh gets skinned on the CPU instead of the GPU. Software skinning is required for blend shapes and to have tangent frames recalculated every frame.</td>
</tr>
<tr>
<td>Linear Skinning GPU</td>
<td>If selected, sends the bone transforms as dual quaternions to the GPU and converts them to matrices on the GPU before blending them together for scaling. For more information, see Enabling Linear Skinning with Geppetto (p. 47).</td>
</tr>
<tr>
<td>Linear Skinning CPU</td>
<td>If selected, converts the bone transforms to matrices on the CPU before sending the matrices to the GPU for skinning. For more information, see Enabling Linear Skinning with Geppetto (p. 47).</td>
</tr>
</tbody>
</table>

**Enabling Linear Skinning with Geppetto**

You can enable linear skinning for animated characters so character skinning uses matrices instead of dual quaternions. Linear skinning can remove some of the artifacts that appear with dual quaternions, such as bulging around joints.

**Note**
Skinning matrices can present their own artifacts, so consider both dual quaternion and linear skinning for your characters.

**To enable linear skinning**

1. In Lumberyard Editor, choose Tools, Legacy/Animation, Geppetto Editor.
2. In the Assets pane, choose a .cdf file with a skin attachment.
3. In the Properties pane, choose Attachments to display the skin attachment options.
4. Choose one of the following options:
   - Linear Skinning GPU – Sends the bone transforms as dual quaternions to the GPU and converts them to matrices on the GPU before blending them together for scaling.
- **Linear Skinning CPU** – Converts the bone transforms to matrices on the CPU before sending the matrices to the GPU for skinning.

**Note**
Both options display identical visual results, but as a best practice, use **Linear Skinning GPU** for better performance.
If you choose both options, **Linear Skinning CPU** overrides **Linear Skinning GPU**.

The following example uses dual quaternion skinning, which is the default if you don't enable linear skinning. Notice the bulging around the joints.
The following example enables linear skinning. Notice that the bulging around the joints is reduced.
Collision Detection and Response

Collision detection and response involves the realistic depiction of attachments that collide with the body of a living character that simulate real-world physics. To do this, a collision proxy is used to approximate parts of a character body such as the legs and torso with a simple geometry shape. Using a collision proxy is more efficient than undertaking all the necessary computation required for collision detection and response with a polygonal mesh.

For information about how to set up a collision proxy attachment, see Proxy (Collision) Attachments (p. 45).

Collision Detection

Collision detection involves checking to see if a blue proxy capsule/sphere overlaps (collides) with a gray proxy lozenge. For pendulums, a blue proxy is always connected to an attachment socket (pivot) at one end.
Collision response is handled by projections. If a dynamic (blue) proxy capsule/sphere collides with an auxiliary (gray) proxy lozenge, the blue proxy is projected or moved away until it no longer overlaps (collides with) the gray proxy lozenge. This means projecting (moving) the blue proxy capsule/sphere perpendicularly from the lozenge surface or rotating it out of the lozenge.

Lumberyard performs two consecutive constraint checks for collision detection. First, the blue proxy capsule/sphere is moved out of the gray proxy lozenge, and second the spring particle or pendulum rod movement is clamped to the shape of the bounding volume: spring ellipsoid, pendulum cone or half-cone, pendulum hinge, or translation projection.

After these two checks, the blue proxy capsule/sphere should be outside of the gray proxy lozenge but inside of the bounding volume. However, if the bounding volume is too small, the collision response may happen successfully only to have the bounding volume push the capsule/sphere back inside the lozenge.

There are four different projection methods used to move proxies to a non-colliding state, depending on the bounding volume, in addition to No Projection:

**Topics**

- Spring Ellipsoid Response (p. 51)
- Pendulum Cone and Half-Cone Response (p. 52)
- Pendulum Hinge Response (p. 52)
- Translational Projection Response (p. 52)

If No Projection is selected, collisions are ignored and no response is initiated.

**Spring Ellipsoid Response**

Selecting Shortvec Translation moves a gray proxy sphere away from a blue proxy lozenge using the shortest distance possible. For springs, only gray proxy spheres (and not capsules) are supported with spring motions.
Pendulum Cone and Half-Cone Response

Selecting **Shortarc Rotation** rotates a gray proxy capsule out of a blue proxy lozenge using the smallest angle possible.

Pendulum Hinge Response

Selecting **Shortarc Rotation** rotates a gray proxy capsule out of a blue proxy lozenge using the shortest direction possible. For hinges, there are only two ways for a capsule to rotate out of a lozenge.

Selecting **Directed Rotation** rotates a gray proxy capsule out of a blue proxy lozenge along the (green) direction of the hinge-plane.

Translational Projection Response

In the case of rotations (**Shortarc Rotation** and **Directed Rotation**), the pivot for a blue proxy capsule must lie outside of a gray proxy lozenge. The pivot is the spherical portion of the capsule that is connected to the attachment socket.

If the capsule pivot lies inside of a lozenge, collisions cannot be resolved and the proxies remain in an overlapping (collided) state. This can occur for secondary animations on characters where the simulation update is triggered after the animation update and it happens that the animation itself moves proxies into each other or creates invalid proxy configurations that break the simulation. To handle these cases, **Translational Projection** type is used, which defines the direction of movement. There are two types of translational projections:

- **Shortvec Translation**
- **Directed Translation**

**Note**

It is important that the new socket is on the same joint where you want to perform the translation and appears in the list of attachments ahead of the pendulum attachment that you want to move out. You can change the order of attachments in Geppetto. This order defines the order of execution, so the translation operation moves the joint out of the proxies before the pendulum attachment is executed.

Selecting **Shortvec Translation** moves a blue proxy capsule out of a gray proxy lozenge along the shortest vector from the surface of a sphere enclosing the joint, where the radius of the sphere is specified. This type should be used in cases where there are only a few lozenges, due to potential unpredictable and undesirable movements.

If an overlap is detected, the sphere is translated out of the lozenge along the shortest vector to the surface. This method of translation is only recommended for setups with just a few lozenges where the results are predictable. Otherwise, issues may arise where the first shortvec translation moves a capsule out of the first lozenge and directly into a second lozenge. These issues are very likely with complex setups where many lozenges are close together or overlap. It can also happen that it projects out in the wrong direction and produces undesired “tunneling” effects.
Choosing Directed Translation moves a blue proxy capsule out of a gray proxy lozenge along either a Translation Axis (defined relative to a joint and socket orientation) in its negative direction, or moves it out relative to a selected Directional Translation Joint, which defines the translation axis between the joint and socket. Optionally, you can select a joint, which forms a translation axis between the location of the joint and the socket.

Both options allow you to specify a capsule in the direction of the translation axis; however, the capsule is always projected out in the predefined direction even if the capsule is behind the lozenge, which makes "tunneling" unlikely.
Lumberyard Legacy Reference
Character Attachments

- **Projection Type**: Directed Translation
- **Directional Translation Joint**: Bip01 Spine2
- **Capsule**: 0.018
- **Pivot Offset**: 0.018

**Available Collision Proxies**
- Spine 1
- Spine 2
- Spine 3
- Rupperarm
- Lupperarm
- Rthighthin
- Lthighthin
Secondary Animations (Simulations)

You can also use sockets to produce realistic movements of joint and face attachments. This type of animation is always a reaction to a primary (character) animation, and are called secondary animations or motion simulations. Such animations can simulate the movement of attached static objects such as weapons and holsters, muscles, and fat.

In addition, it is also possible to create complex setups to simulate the motions of swinging hair braids, tentacles, chains, ropes, necklaces, clothing, and other loose or dangling objects on a character. Chains can have branching strings and different physical properties for each link.

However, such motions are just approximations of real-world physical movements. In Lumberyard, the physical properties of springs and pendula are used to approximate (simulate) the physical movement of dangling or swinging objects attached to characters.

- **Pendulum**: A bob connected to a rigid rod that experiences simple harmonic motion as it swings back and forth. The equilibrium position of an unconstrained pendulum is hanging directly downward. The swing is specified by physical parameters such as stiffness and stiffness target, and movement is constrained by cone, half cone, or hinge plane bounding volumes.

- **Spring Ellipsoid**: A bob connected to an elastic rod. Unlike a helical spring, a spring ellipsoid can stretch in any direction. The movement of the spring is constrained to by sphere, ellipsoid, half sphere, flat plane, or line bounding volumes.

Moving springs and pendula have different motion bounding volumes that constrain the movement of objects attached to characters.

While the type, size, and shape of the attachment has no impact on its actual motions, it does determine which type of simulation is selected as the movements of a corresponding real-world physical object must be simulated. In this way, the socket and attached object realistically react to the movements of the character.

In addition, because moving attachments may collide with the character, this must be accounted for. For more information, see Proxy (Collision) Attachments (p. 45) and Collision Detection and Response (p. 50).

**Topics**
- Pendulum Cone Simulation (p. 56)
- Pendulum Half-Cone Simulation (p. 57)
- Pendulum Hinge Simulation (p. 58)
- Spring Ellipsoid Simulation (p. 59)
**Pendulum Cone Simulation**

- Redirect to Joint
- Debug Setup
- **Debug Text**
- Activate Simulation

- **Simulation FPS**: 10
- **Simulation Axis**: 0, 0.5, 0
- **Mass**: 1.0
- **Gravity**: 9.81
- **Damping**: 1.0
- **Joint Spring**: 0
- **Spring Target**: 0, 0
- **Cone Angle**: 45.0
- **Hinge Rotation**: 0
- **Pivot Offset**: 0, 0, 0
- **Capsule**: 0, 0
- **Projection Type**: No Projection
- **Hidden**
- **Physicalized Rays**
- **Physicalized Collisions**
Pendulum Half-Cone Simulation

Simulation

- Redirect to Joint
- Debug Setup
- Debug Text
- Activate Simulation

Simulation FPS: 10
Simulation Axis: 0 0.5 0
Mass: 1
Gravity: 9.81
Damping: 1
Joint Spring: 0
Spring Target: 0 0
Cone Angle: 45
Hinge Rotation: 0
Pivot Offset: 0 0 0
Capsule: 0 0
Projection Type: No Projection
Pendulum Hinge Simulation

This feature is identical for both spring and pendula simulations. Pivot Offset allows you to offset the location of the attached render object. Note that this is purely a visual feature with no impact on the simulation itself and only adds an offset to the attached object at the rendering stage. Adding or changing an offset value doesn't change the position of the socket; it only renders the attached geometry at another location that can be outside of the bounding volume.

By default, it is the pivot of the model (offset = 0,0,0) and those three values are an x,y,z axes offset that translates the rendered geometry in the direction of the socket axes.

If Redirect to Joint is also enabled, then the pivot offset changes the location of the joint and all its children, as discussed next.

Redirect to Joint

If enabled, the relative motion of the simulated object is transferred to the joint that it is attached to, which means that the relative motion of the pendulum is added to the joint. So as long as the pivot offset is (0,0,0) then we only modify the orientation of the joint and this moves all vertices that are part of the mesh and weighted to this joint.
Spring Ellipsoid Simulation

Pivot Offset

This feature is identical for both spring and pendula simulations. Pivot Offset allows you to offset the location of the attached render object. Note that this is purely a visual feature with no impact on the simulation itself and only adds an offset to the attached object at the rendering stage. Adding or changing an offset value doesn’t change the position of the socket, it only renders the attached geometry at another location that can be outside of the bounding volume.

By default, it is the pivot of the model (offset = 0,0,0) and those three values are an x,y,z axes offset that translates the rendered geometry in the direction of the socket axes.

If Redirect to Joint is also enabled, then the pivot offset changes the location of the joint and all its children, as discussed next.

Redirect to Joint

If enabled, the relative motion of the simulated object is transferred to the joint that it is attached to, which means that the distance between the spring particle and the joint is added together. For spring
simulations, only the translation of the joint is changed, which moves all vertices that are part of the mesh and weighted to the joint.

**Animating Characters**

Skeleton-based animation is the most flexible animation technique used today, and includes playback and blending of animation data as well as IK-based poses. Procedural algorithms like CCD-IK, analytic IK, example-based IK, or physical simulations are all used to augment pre-authored animations. To provide realism when combining artificial and captured animations, a warping technique preserves the style and content of the base motion.

However, skeleton-based animation is not the ideal solution for animating muscles and tendons of the human body or face. Although it is possible to use skeleton-based animation for this, the number of joints involved is high and animation setup is difficult.

Generally, the combination of morph-based animation along with skeletal-based animation provides the greatest flexibility. The number of vertices that change in each morph target is very limited and the targets can be clearly defined. Morph targets are ideal for creating facial animations. Morph-based animation can even be used to generate entire animation sequences.

At the highest level, you can use Flow Graph, Lua scripts, or C++ code to request character animations. These methods invoke the Mannequin system, which in turn invokes the core Lumberyard animation system for animation clips, animation events, and procedural clips. Procedural clips can include IK, ragdoll, sounds, particle effects, and game logic.

Geppetto is a tool used to add character attachments, preview animations, and test blending features. It provides a visual interface to the underlying animation system.

You can add character .cdf and geometry .cgf assets to the cinematic cutscene animations in the Track View editor.

**Topics**

- Types of Character Animations (p. 60)
- Character Animation Files (p. 61)
- Using Geppetto to Create a Basic Chrparams File (p. 63)
- Chrparams File Elements (p. 68)
- Character Skeletons (p. 72)
- Importing Character Animations (p. 72)
- Compressing Character Animations (p. 73)
- Working with Additive Animations (p. 74)
- Character Animation Layers (p. 75)
- Working with Blend Shapes (Morphs) (p. 77)
- Working with Blend Spaces (Bspaces) (p. 78)
- Animation Events (p. 80)
- Locomotion Locator Animation Best Practices (p. 81)
- Streaming Character Animations (p. 81)

**Types of Character Animations**

You can produce three major types of animation in Lumberyard:
Cutscene Animations

Cutscenes are cinematic sequences in a game that involve no gameplay. Also known as linear or cinematic animation, cutscene animations are the easiest animation to create, as the animator controls every aspect. Camera angle, lighting, keyframes, and character pose are all fixed. You create cutscene animations with the Track View editor.

Scripted Flow Graph Animations

More complex than cutscene animations are scripted animations in which characters follow a predefined path. The quality is such that it appears to be interactive, but it is not. Characters cannot engage with, or respond to, the player.

You can create scripted animations using animation Flow Graph nodes and can also include AI nodes for more complicated animations. An example would be a character who changes his walking gait over uneven or hilly terrain, or to avoid a vehicle that is in the line of the walking path.

You can use Flow Graph to start and stop animations, trigger animations based on time, synchronize two animations, and coordinate multiple animations based on various parameters.

Interactive Animations

The most complex character animation to create are fully interactive, nonlinear animations where characters respond automatically to their environment, other characters, player inputs, AI behaviors, and other in-game variables. It is common to have a character perform multiple movements and tasks simultaneously, displaying different emotions, and respond differently to different events.

In such an environment, character movements and actions are unpredictable. A crucial feature of interactive animation involves the automatic synthesis of high quality character motions and good AI rules for behavior based on a variety of different game events, all while keeping performance high and asset count as low as possible.

Interactive animations fall into two categories: player controlled and AI controlled.

In player-controlled animations, the player determines the movement and all other actions of the character; the animation system takes the player input and translates it on the fly to skeleton movements using procedural and data-driven methods. For player control, high responsiveness is a key feature.

In AI-controlled animations, the AI system controls the movement and actions of the character. All motion and behaviors are dictated based on a series of rules and parameters that defines a character’s actions in response to in-game events. These actions are not fully predictable as there are an almost unlimited number of different game permutation possibilities.

To help you achieve high quality interactive character animations, Lumberyard provides the following tools:

- **Geppetto** – Lower level system that manages short animation clips, poses, procedural and parameterized movements, transitions, and layers. For more information, see Using Geppetto (p. 33).
- **Mannequin Editor** – High-level system that manages animation variations, transitions, sequences, blends, layers, and procedural logic. For more information, see Using Mannequin Editor (p. 86).

Character Animation Files

To create character animation files, you start by animating character skeletons and hierarchies in a DCC tool. You then use your DCC tool to export these elements to the intermediate \*.i_caf file format. They are then compressed and optimized to the \*.caf file format before Lumberyard can use them.
Lumberyard's animation system uses the following files to create animations for your characters.

Character Asset File (*.chr)

The character used for animations is defined in a .chr file. For animation, the two important aspects of a character are the morph targets and the skeleton.

Character Definition File (*.cdf)

Characters are usually combinations of a primary model and several attachments. In particular, the head is often considered a skin attachment that is a separate model attached to the body. This composite model is defined in the .cdf file and contains a reference to the .chr file and its attachments.

Intermediate Character Animation File (.i_caf)

The intermediate character animation file contains the animated bone data for a specific character. This file can be used with multiple characters with similar bone structures. The file is created by a DCC tool and stores animation data in un compressed format. It is usually used with a skinned mesh.

Animation Settings File (.animsettings)

The animation settings file contains per-animation compression settings. This is a sidecar file that is stored next to the .i_caf file and describes how it should be compiled by the Asset Pipeline. This file is created using Geppetto for importing animations.

Skeleton Alias File (SkeletonList.xml)

This file provides a table that maps skeleton aliases used in the .animsettings file to skeleton file names. This file contains skeleton structure information that is needed during animation compression.

Character Animation File (*.caf)

Assets, such as bones, are stored in .caf files. Because they are considered on demand assets, these files are streamed in and out as needed. This file is the compressed version of the intermediate .i_caf file and uses lossy compression. Character animation files are created by Lumberyard Editor during the asset build, and are loaded by the game at runtime.

Character Parameters File (*.chrparams)

Skeletal character parameters are defined in the XML .chrparams file. This file has the same name as the .chr character file to which it refers.

Animation Database (.dba)

A .dba file consists of multiple animations (character, player, AI, weapons) that are streamed in and out together. These files are typically smaller and take up less memory than individual animations (.caf files). Single .caf files are no longer needed unless they are on-demand assets.

If an animation is in a .dba file, it will not be available anymore as an individual .caf file. If the game tries to play one of these animations, the database containing that animation loads instead. As this can take a while, make sure the .dba is preloaded.

When two animations in the same .dba file have exactly the same animation for a joint, the data for that animation is stored once. This can provide significant memory savings.

The .dba files are created by the Resource Compiler after compressing the individual animations (.caf), according to the DbaTable.xml file. The .dba file must be defined in the .chrparams file.

Typical animations that get stored in the .dba include:

• Animations that need to be individually loaded and unloaded.
• Animations that need to be accessed once on demand, such as track view (cinematic) animations. These animations are preloaded a couple of seconds before starting.

**Note**
Aimposes, Lookspace, .bspace, and .comb files cannot be stored in a .dba database.

**Animation Database Table (DbaTable.xml)**

The animation database table contains a list of .dba files, which the resource compiler uses to determine which .caf animations to put in which .dba files. Here is an example:

```xml
<DBAs>
  <DBA Path="Animations\human\male\hits_1p.dba">
    <Animation Path="Animations\human\male\hits\1p\stand_tac_hit_generic_add_1p_01.caf"/>
    <Animation Path="Animations\human\male\hits\1p\stand_tac_hit_knockDown_1p_01.caf"/>
    <Animation Path="Animations\human\male\hits\1p\stand_tac_idle_reactExplosion_3p_01.caf"/>
  </DBA>
  <DBA Path="Animations\human\male\locomotion.dba">
    <Animation Path="Animations\human\male\locomotion\kneel\kneel_tac_AimPoses_idle_01.caf"/>
    <Animation Path="Animations\human\male\locomotion\kneel\kneel_tac_death_01.caf"/>
    <Animation Path="Animations\human\male\locomotion\kneel\kneel_tac_idle_01.caf"/>
    <Animation Path="Animations\human\male\locomotion\kneel\kneel_tac_stepRotate_90_lft_01.caf"/>
  </DBA>
</DBAs>
```

**Animation Events Database (.animevents)**

This database stores a list of assets with timed event markups. For example, it might store footstep sounds. You use the Geppetto to create this database, which gets mapped to the .chrparams file.

**Blend Space (.bspace)**

Blend spaces (Bspaces) define how multiple animation assets are blended together. Blend spaces are parameterized at runtime with movement parameters such as movement speed, movement direction, turning angle, or slope.

**BlendSpace Combination (.comb)**

This file combines multiple blend spaces into one, usually of a higher order, and represents a multidimensional blend space.

**Group Files (*.grp)**

Group files are exported animation sequences in XML format that are used for track view animation sequences. Data stored in a sequence includes everything from audio positions to skeletal animations and camera paths used.

**Using Geppetto to Create a Basic Chrparams File**

Geppetto allows you to set up mapping for animations, animation events, database animations, bounding box extensions, bounding box includes, joint LODs, and IK definitions. At a minimum you can use the .chrparams file to map animations to a specific character skeleton. You can also include other .chrparams settings. This topic provides details about each property and how to apply them to a .chrparams file.

**Includes and Included Animation Set Filter**

You can include another .chrparams Animation Sets for the currently loaded .chrparams file to use. Animation set data for the included .chrparams file is listed under the **Included Animation Set**
**Filter** property, which only displays read-only data. This may be useful if you have characters that can share animations because their skeletons are identical, but one particular version has a unique subset of animations.

**To add an Includes entry to a .chrparams file using Geppetto**

1. In Lumberyard Editor, click **Tools, Geppetto**.
2. In **Geppetto**, in the **Assets** panel, expand **Skeletons** and navigate to a character's skeleton (*.chrparams).
3. Select the .chrparams file.
4. In the **Properties** panel, click the **Includes** drop-down list and select **Add**.
5. Click the folder and assign a .chrparams file to include.
6. The **Included Animation Set Filter** field adds new entries based on what exists within the .chrparams file that you added to the **Includes** property. These properties are read-only and can only be modified by loading the original .chrparams file to which the properties belong.
7. Add additional .chrparams files as needed.
8. In the **Properties** panel, click **Save** (disk icon) to save your changes to the .chrparams file.

**Animation Set Filter**

You must assign an animation set (path) in order for a character's skeleton to recognize where animations are located to use. You can apply multiple filters if there are several animation directories. Further specify each animation set (path) filter that you have applied by setting naming filters or file extension filters.

**To add an animation set to a .chrparams file using Geppetto**

1. In Lumberyard Editor, click **Tools, Geppetto**.
2. In **Geppetto**, in the **Assets** panel, expand **Skeletons** and navigate to a character's skeleton (*.chrparams).
3. Select the .chrparams file.
4. In the **Properties** panel, click the **Animation Set Filter** drop-down list and select **Add**.
5. The **Animation Set Filter** field adds an empty property field with additional fields below.
6. Click the folder next to the empty field to set the directory where the animations for the character's skeleton are located. By default, there are three filters under the path to search for any named animation with the .caf, .bspace, and .comb extensions in the assigned directory and subdirectories within the path.

**Note**

The left field represents the alias name of the asset to use in-game. The asterisk (*) in this field represents a use-default-name or a pass-through. In the right field, the first asterisk (*) in */* .caf represents including all subfolders. The second asterisk (*) includes all found .caf files. For example, setting **run_loop** in the left field and **chicken_run.caf** in the right field will assign the alias **run_loop** to the **chicken_run.caf**. Engine systems would then use **run_loop** as the animation name.

7. (Optional) Add additional filters to the assigned animation directory path by clicking the drop-down menu next to the folder icon that allowed you to assign your animation directory path.
8. In the **Properties** panel, click **Save** (disk icon) to save your changes to the .chrparams file.
9. You can add animation directories as needed and modify the subfields if you need to set specific filters for your animations.
10. When you load a .caf file in Geppetto using the skeleton, you will see the animations from the assigned directory listed under **Animations** in the **Assets** panel.
**Animation Events**

You must map an `.animevents` file to the `.chrparams` file in order for animation events to be applied to the animations that are available for a character's skeleton. Only one `.animevents` file can be assigned per `.chrparams` file.

You can do either of the following:

- Manually create an `.animevents` file and assign it to the `.chrparams` file.
- Use Geppetto to automatically create the `.animevents` file and assign it to the `.chrparams` file. At least one animation must be available to play on the character's `.cdf`.

**To manually add an AnimEvents path to a `.chrparams` file**

1. In Lumberyard Editor, click **Tools, Geppetto**.
2. In **Geppetto**, in the **Assets** panel, expand **Skeletons** and navigate to a character's skeleton (*.chrparams).
4. In the **Properties** panel, click the folder next to the **Events** field.
5. Select the directory where the `.animevents` file exists for the character's skeleton.
   
   **Note**
   
   Only one `.animevents` file can be assigned per `.chrparams` file.
6. In the **Properties** panel, click **Save** (disk icon) to save your changes to the `.chrparams` file.

**To automatically create an `.animevents` file and assign it to a `.chrparams` file**

1. In Lumberyard Editor, click **Tools, Geppetto**.
2. In **Geppetto**, in the **Assets** panel, expand **Characters** and double-click the character for which you want to create an `.animevents` file.
3. In the **Assets** panel, under **Animations**, select a character animation to load.
4. In the **Playback** panel, double-click anywhere in the timeline to add an AnimEvent.
5. In the **Properties** panel, under **Animation Events**, change the settings and properties of the AnimEvent.
6. In the **Properties** panel, click **Save** (disk icon) to save your changes and create the `.animevents` file.
7. In the **Assets** panel, expand **Skeletons** and select the character's skeleton (*.chrparams).
8. In the **Properties** panel, verify that the `.animevents` file appears in the **Events** field.
   
   **Note**
   
   The `.animevents` file is created in the same location as the animation with the added AnimEvent. Ensure that you check in this file if you are using source control.
9. In the **Properties** panel, click **Save** (disk icon) to save the `.animevents` file to the `.chrparams`.

**Database Animations (DBA) Path**

You must assign the path for the directory that contains multiple database animations (.dba) files to the skeleton's `.chrparams` file in order for a character's skeleton to read the animations.

**To add a DBA path to a `.chrparams` file using Geppetto**

1. In Lumberyard Editor, click **Tools, Geppetto**.
2. In **Geppetto**, in the **Assets** panel, expand **Skeletons** and navigate to a character's skeleton (*.chrparams).

---

65
3. Select the .chrparams file.
4. In the Properties panel, click the folder next to the DBA Path field.
5. Select the directory where the .dba files exist.
6. In the Properties panel, click Save (disk icon) to save your changes to the .chrparams file.

**Individual Database Animations (DBAs)**

You must assign the path of the database animation (.dba) file to the skeleton's .chrparams file in order for a character's skeleton to read animations from a single .dba file.

**To add an individual DBA path to a .chrparams file using Geppetto**

1. In Lumberyard Editor, click Tools, Geppetto.
2. In Geppetto, in the Assets panel, expand Skeletons and navigate to a character's skeleton (*.chrparams).
3. Select the .chrparams file.
4. In the Properties panel, click the Individual DBAs drop-down list and select Add.
5. Click the folder and assign an individual DBA. You can also select the Persistent check box to keep the DBA in memory after the character has loaded.
6. In the Properties panel, click Save (disk icon) to save your changes to the .chrparams file.

**Bounding Box Extension**

You can extend a skeleton's bounding box based on the size of the skeleton.

**To add a bounding box extension to a .chrparams file using Geppetto**

1. In Lumberyard Editor, click Tools, Geppetto.
2. In Geppetto, in the Assets panel, expand Skeletons and navigate to a character's skeleton (*.chrparams).
3. Select the .chrparams file.
4. In the Properties panel, expand Bounding Box Extension.
5. Set the positive (Pos) and negative (Neg) values for the XYZ coordinates to extend the bounding box of the skeleton.
6. Preview the extended bounding box by loading a .cdf file that uses the skeleton's .chrparams. Click Display Options, Skeleton, and select the Bounding Box check box to see your bounding box on the character.
Bounding Box Include

A skeleton's bounding box is defined by the location of the joints within the skeleton. You can select which joints in the skeleton define the size of the bounding box. By default, all joints are included.

To modify the bounding box include in a .chrparams file using Geppetto

1. In Lumberyard Editor, click Tools, Geppetto.
2. In Geppetto, in the Assets panel, expand Skeletons and navigate to a character's skeleton (*.chrparams).
3. Select the .chrparams file.
4. In the Properties panel, expand Bounding Box Include.
5. Select the skeleton joints to include for calculating the skeleton's bounding box.
6. In the Properties panel, click Save (disk icon) to save your changes to the .chrparams file.
Joint LOD

You can assign a skeleton's joints with an LOD so that certain joints stop being used after a certain LOD, based on camera distance. This can help cut down performance cost of reading all joint animation data. By default, all joints are enabled and do not use LOD.

**Note**
Joint LODs are treated globally to the skeleton joints. You can choose to include only specific joints on an individual animation when importing the animation.

To add a joint LOD to a `chrparams` file using Geppetto

1. In Lumberyard Editor, click **Tools, Geppetto**.
2. In **Geppetto**, in the **Assets** panel, expand **Skeletons** and navigate to a character's skeleton (*.chrparams).
3. Select the `.chrparams` file.
4. In the **Properties** panel, click the **Joint LOD** drop-down list and select **Add**.
5. For the LOD 1 entry, select the skeleton joints to enable.
6. Add additional entries and modify the incremental joint LODs.
7. In the **Properties** panel, click **Save** (disk icon) to save your changes to the `.chrparams` file.

IK Definition

The IK definition provides access to various IK types such as Limb IK, Aim IK, Look IK, Feet Lock IK, and Recoil IK. You can enable the IK types for a character's skeleton using the `.chrparams` properties.

To add an IK definition to a `.chrparams` file using Geppetto

1. In Lumberyard Editor, click **Tools, Geppetto**.
2. In **Geppetto**, in the **Assets** panel, expand **Skeletons** and navigate to a character's skeleton (*.chrparams).
3. Select the `.chrparams` file.
4. In the **Properties** panel, expand **IK Definition**.
5. Select the check boxes for the IK definitions that you need for your character. If there are additional settings, expand the IK definition.

**Note**
For information about setting up specific IK definitions, see [Using Inverse Kinematics (IK)](p. 128).

6. In the **Properties** panel, click **Save** (disk icon) to save your changes to the `.chrparams` file.

Chrparams File Elements

All parameters for a character in Lumberyard is stored in various element sections of the `.chrparams.xml` file. You can use any text editor to edit this XML file.

Animations

The `.chrparams` file contains a single `<AnimationList>` element. This element lists all animation asset files that the character uses. See the following example.

```xml
<AnimationList>
  <Animation name="$AnimEventDatabase" path="animations\human\male\events.animevents"/>
  <Animation name="$Include" path="animations\human\male\male.chrparams"/>
  <Animation name="$TracksDatabase" path="animations\human\male\hits.dba"/>
</AnimationList>
```
Bone LODs

The `.chrparams` file contains a single `<Lod>` element section, which lists all joints that the character uses. See the following example:

```xml
<Lod>
  <JointList level="0">
    <Joint name="Bip01 Pelvis"/>
    <Joint name="Bip01 Spine"/>
  </JointList>
  <JointList level="1">
    <Joint name="weapon_bone"/>
    <Joint name="joint_12"/>
  </JointList>
</Lod>
```

IK Definition

The `.chrparams` file contains a single `<IK_Definition>` element section, which defines the joint that are used for the different IK methods, such as AimIK, LookIK, LimbIK and Animation-Driven IK.

Limb

The `.chrparams` file contains a single `<LimbIK_Definition>` element section within `<IK_Definition>`. This section lists all the joints that are used for Limb IK, along with the root bone, end effector, and solver. See the following example section:

```xml
<IK_Definition>
  <LimbIK_Definition>
    <IK EndEffector="Bip01 R Hand" Handle="RgtArm01" Root="Bip01 R UpperArm" Solver="2BIK"/>
    <IK EndEffector="Bip01 L Hand" Handle="LftArm01" Root="Bip01 L UpperArm" Solver="2BIK"/>
    <IK EndEffector="Bip01 R Foot" Handle="RgtLeg01" Root="Bip01 R Thigh" Solver="2BIK"/>
    <IK EndEffector="Bip01 L Foot" Handle="LftLeg01" Root="Bip01 L Thigh" Solver="2BIK"/>
  </LimbIK_Definition>
</IK_Definition>
```

Anim Driven

The `.chrparams` file contains a single `<Animation_Driven_IK_Targets>` element section within `<IK_Definition>`. This section lists all joints used for Animation-driven IK, along with target bones, blend bones, and weights. See the following example section:

```xml
<IK_Definition>
  <Animation_Driven_IK_Targets>
    <ADIKTarget Handle="LftArm01" Target="Bip01 Chin_IKTarget" Weight="Bip01 Chin_IKBlend"/>
  </Animation_Driven_IK_Targets>
</IK_Definition>
```
Foot Lock

The .chrparams file contains a single <FootLock_Definition> element section within <IK_Definition>. This block lists all joints used for foot step alignment and lock effects. See the following example section:

```xml
<IK_Definition>
  <FootLock_Definition>
    <RIKHandle Handle="RgtLeg01"/>
    <LIKHandle Handle="LftLeg01"/>
  </FootLock_Definition>
</IK_Definition>
```

Recoil

The .chrparams file contains a single <Recoil_Definition> element section within <IK_Definition>. This block lists all weapon joints and impact joints used for recoil effects, along with weights and delay times. See the following example section:

```xml
<IK_Definition>
  <Recoil_Definition>
    <RIKHandle Handle="RgtArm01"/>
    <LIKHandle Handle="LftArm01"/>
    <RWeaponJoint JointName="weapon_bone"/>
    <ImpactJoints>
      <ImpactJoint Arm="3" Delay="0.3" Weight="0.2" JointName="Bip01 Pelvis"/>
      <ImpactJoint Arm="3" Delay="0.2" Weight="0.3" JointName="Bip01 Spine"/>
      <ImpactJoint Arm="3" Delay="0.1" Weight="0.5" JointName="Bip01 Spine1"/>
      <ImpactJoint Arm="3" Delay="0.0" Weight="1.0" JointName="Bip01 Spine2"/>
      <ImpactJoint Arm="3" Delay="0.0" Weight="1.0" JointName="Bip01 Spine3"/>
      <ImpactJoint Arm="3" Delay="0.0" Weight="1.0" JointName="Bip01 Neck"/>
      <ImpactJoint Arm="3" Delay="0.10" Weight="0.10" JointName="Bip01 R Thigh"/>
      <ImpactJoint Arm="3" Delay="0.05" Weight="0.05" JointName="Bip01 R Calf"/>
      <ImpactJoint Arm="3" Delay="0.10" Weight="0.10" JointName="Bip01 L Thigh"/>
      <ImpactJoint Arm="3" Delay="0.05" Weight="0.05" JointName="Bip01 L Calf"/>
      <ImpactJoint Arm="2" Delay="0.0" Weight="1.0" JointName="Bip01 R Clavicle"/>
      <ImpactJoint Arm="2" Delay="0.00" Weight="0.50" JointName="Bip01 R UpperArm"/>
      <ImpactJoint Arm="1" Delay="0.01" Weight="0.7" JointName="Bip01 L Clavicle"/>
      <ImpactJoint Arm="1" Delay="0.00" Weight="0.50" JointName="Bip01 L UpperArm"/>
    </ImpactJoints>
  </Recoil_Definition>
</IK_Definition>

Look

The .chrparams file contains a single <LookIK_Definition> element section within <IK_Definition>. This block lists all joints used for Look IK, along with eye attachments, limits, and rotations. See the following example section:

```xml
<IK_Definition>
  <LookIK_Definition>
    <LEyeAttachment Name="eye_left"/>
    <REyeAttachment Name="eye_right"/>
    <DirectionalBlends>
```

70
Lumberyard Legacy Reference
Animating Characters

```
<Joint AnimToken="LookPoses" ParameterJoint="Bip01 Look" StartJoint="Bip01 Look"
ReferenceJoint="Bip01 Pelvis"/>
</DirectionalBlends>

<RotationList>
<Rotation Additive="1" Primary="1" JointName="Bip01 Pelvis"/>
<Rotation Additive="1" Primary="1" JointName="Bip01 Spine"/>
<Rotation Additive="1" Primary="1" JointName="Bip01 Spine1"/>
<Rotation Additive="1" Primary="1" JointName="Bip01 Spine2"/>
<Rotation Additive="1" Primary="1" JointName="Bip01 Spine3"/>
<Rotation Additive="0" Primary="1" JointName="Bip01 Neck"/>
<Rotation Additive="0" Primary="1" JointName="Bip01 Head"/>
<Rotation Additive="0" Primary="1" JointName="Bip01 Look"/>
<Rotation Additive="0" Primary="0" JointName="def_r_brow_A"/>
<Rotation Additive="0" Primary="0" JointName="def_r_brow_B"/>
<Rotation Additive="0" Primary="0" JointName="def_r_brow_C"/>
<Rotation Additive="0" Primary="0" JointName="def_r_upperEyeLid"/>
<Rotation Additive="0" Primary="0" JointName="def_r_lowerEyeLid"/>
<Rotation Additive="0" Primary="0" JointName="def_l_brow_A"/>
<Rotation Additive="0" Primary="0" JointName="def_l_brow_B"/>
<Rotation Additive="0" Primary="0" JointName="def_l_brow_C"/>
<Rotation Additive="0" Primary="0" JointName="def_l_upperEyeLid"/>
<Rotation Additive="0" Primary="0" JointName="def_l_lowerEyeLid"/>
</RotationList>

<PositionList>
<Position Additive="1" JointName="Bip01 Pelvis"/>
<Position Additive="0" Primary="0" JointName="def_r_brow_A"/>
<Position Additive="0" Primary="0" JointName="def_r_brow_B"/>
<Position Additive="0" Primary="0" JointName="def_r_brow_C"/>
<Position Additive="0" Primary="0" JointName="def_r_upperEyeLid"/>
<Position Additive="0" Primary="0" JointName="def_r_lowerEyeLid"/>
<Position Additive="0" Primary="0" JointName="def_l_brow_A"/>
<Position Additive="0" Primary="0" JointName="def_l_brow_B"/>
<Position Additive="0" Primary="0" JointName="def_l_brow_C"/>
<Position Additive="0" Primary="0" JointName="def_l_upperEyeLid"/>
<Position Additive="0" Primary="0" JointName="def_l_lowerEyeLid"/>
</PositionList>

</LookIK_Definition>
</IK_Definition>

Aim
The .chrparams file contains a single <AimIK_Definition> element section within <IK_Definition>. This block lists all joints required for Aim IK, along with positions, rotations, and procedural adjustment joints. See the following example section:

```
<AimIK_Definition>
  <DirectionalBlends>
    <Joint AnimToken="AimPoses" ParameterJoint="weapon_bone" StartJoint="Bip01 R UpperArm" ReferenceJoint="Bip01"/>
  </DirectionalBlends>
  <RotationList>
    <Rotation JointName="Bip01 Spine" Primary="1" Additive="0"/>
  </RotationList>
</AimIK_Definition>
```
Use the following procedure to add a character skeleton to the SkeletonList.xml file.

To add a character skeleton to the skeleton list

1. In Lumberyard Editor, click Tools, Geppetto.
2. In Geppetto, in the Assets panel, expand Compression (Animations) and click Skeleton List.
3. In the Properties panel under Aliases, make sure the skeleton .chr file is in the list. If not, do the following:
   a. Click the number button next to Aliases and click Add.
   b. Click the folder icon next to the new entry, then select a suitable skeleton.
   c. Name the added skeleton alias - this name is used to refer to the skeleton.

Skeleton Aliases

This file provides a table that maps skeleton aliases used in the .animsettings file to skeleton file names. This file contains skeleton structure information that is needed during animation compression.

Importing Character Animations

You can easily import character animations using Geppetto. The character's skeleton needs to be part of the skeleton list before you can start importing animations. Have your character loaded in Geppetto before following the steps below.
To import character animations

1. In Lumberyard Editor, click **Tools, Geppetto**.
2. In Geppetto, in the **Assets** panel, expand **Animations** and select the animation to import. All unimported animations become unavailable.
3. In the **Properties** panel, click **Import**.

   **Note**
   You may need to select **Skeleton Alias** in the menu in the event that the loaded character could not be matched to the skeleton alias.

Compressing Character Animations

For best results, try to employ character assets that use the least amount of memory but are animated at the highest possible quality. An uncompressed animation contains a key for every frame in the animation and for each joint that has been exported. The goal is to reduce the amount of joints and keys to minimize the size. There are separate channels for rotation keys and position keys per joint.

For maximum compression, remove from the animation any joints that don't contribute much to the animation. To know whether a joint contributes to an animation, use the Resource Compiler, which determines how much the joint moves during the animation and compares it to the provided epsilon values. If the joint moves less than what the epsilon specifies, the keys will be removed for the joint. Use higher epsilon values to remove more joints. Use **Position Epsilon** for the position channel and **Rotation Epsilon** for the rotation channel.

Removing Joints Automatically

The two epsilon values are global values for the entire animation. Additive animations have smaller movements, so small values are used for the epsilon values.

Either all the keys are retained for a channel (position and rotation), or they are deleted.

To remove joints automatically

1. In Geppetto, in the **Properties** panel, expand **Compression, Controller Removal Threshold**.
2. Change the values for **Position Epsilon** and **Rotation Epsilon** as needed.

Removing Joints Manually

By default, each joint uses two epsilon values to determine whether the joint is removed.

To remove individual joints manually

1. In Geppetto, in the **Properties** panel, expand **Compression, Per-Joint Settings** and then select the check box next to the joint to delete it. Both the **Position** and **Rotation** channels are removed for the selected joint.
2. Enter a value in the box next to the joint to change the multiplier that is applied to the compression value. By default this value is 1.

Animation Tags

Each animation can have a list of tags associated with it. You can use tags to accomplish the following:

- Select animations that have to go into a specific DBA table by means of an animation filter.
- Apply compression to a group of animation files by means of compression presets.
Tags are located in the **Properties** panel when you select an animation in the **Assets** panel.

To add a new tag, click the number beside **Tags** and click **Add**.

### Animation Filters

Use an animation filter to choose a set of animation files for specific DBA or compression preset. An animation filter is defined as a tree of condition nodes.

DBA files are bundles of animations that can be streamed in and out as one. They are typically smaller and take up less memory than individual animations. DBAs are created using the same filters as compression presets. You can define a combination of criteria such as location, name, or tags to select animations for a specific DBA.

DBA descriptions are saved to `Animations/DBATable.json`. The resource compiler uses this `.json` file at build time to create the actual DBA files. The DBA Table can be found under **Compression (Animations/)** in the **Assets** panel.

### DBA Table Options

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty filter</td>
<td>No conditions applied</td>
</tr>
<tr>
<td>And</td>
<td>Succeeds when all of the child conditions succeed.</td>
</tr>
<tr>
<td>Or</td>
<td>Succeeds when at least one of the child conditions succeeds.</td>
</tr>
<tr>
<td>In folder</td>
<td>Checks whether animation is located within a specific directory.</td>
</tr>
<tr>
<td>Has tags</td>
<td>Checks whether animation has all of the listed tags. Tags are stored in ANIMSETTINGS and can be set in Animation Properties.</td>
</tr>
<tr>
<td>Contains in name</td>
<td>Checks for a substring within an animation name.</td>
</tr>
<tr>
<td>Contains in path</td>
<td>Checks whether animation is located within a specific file path.</td>
</tr>
<tr>
<td>Skeleton alias</td>
<td>Checks whether animation uses a specific skeleton alias. Skeleton aliases are defined in the skeleton table.</td>
</tr>
</tbody>
</table>

### Compression Presets

You can use compression presets to apply the same set of compression rules to multiple animations at once. Presets are listed under **Compression (Animations/)** in the **Assets** panel.

Each compression preset entry defines a filter that can match animations according to a certain filter. Filter criteria can include a folder, file name, or tags. You can use logical operations to combine these criteria into a complex condition like "in folder and doesn't contain specific tag but has substring in name." When multiple presets match the same animation, only the first one is used. You can preview which compression setting entry was applied to animation in the **Properties** panel by selecting a specific animation in the **Assets** panel.

### Working with Additive Animations

Additive animations are animations that can be added as layers on top of a base animation. The additive animation is usually a partial-body animation, so it can be applied to a base full-body animation without interfering with joint controllers and other important parts of the base animation. With additive animations, you can reuse the same full body-animations and add lots of variation to it.
An additive animation preserve the underlying animation and style and as such is great for adding poses and animations to the upper body. Since the underlying animations are not overwritten, this can reduce the overall asset count greatly, add a lot of variation to the animations, and reduce the monotonous look.

For example, you can use additive animations for breathing, looking around, flinching, and posture changes. To prevent foot sliding, additive animations cannot modify bones below the character's hips.

You start an additive animation like a regular animation. Lumberyard automatically recognizes it after it has been processed by the resource compiler.

Creating Additive Animations

To create an additive animation, you start with a typical base pose and then animate only those bones and other parts that you want to include in the additive animation. The first frame (frame 0) is the base pose and the rest of the animation becomes the delta. Bones that do not differ from the base pose are not used. The resource compiler subtracts the first frame during export; it is not part of the final animation.

Importing Additive Animations

To import an additive animation, select Additive check box for the .i_caf animation in the Properties panel in Geppetto.

Testing Additive Animations

You can test an additive animation in just a few steps

To test an additive animation
1. In Lumberyard Editor, click Tools, Geppetto.
2. Click File, Open and load the applicable character .cdf file.
3. Make sure a full body animation is playing on the first animation layer under Animation Layers in the Scene Parameters panel.
4. Add a new animation layer in the Scene Parameters panel by clicking on the number next to Animation Layers, and then click Add.
5. Select an additive animation from the Animation list in the Assets panel to add it to the new animation layer. Adjust the weights of the additive animation as needed by changing the value (0 to 1) next to the new animation layer.

Character Animation Layers

By layering animations, you can apply an animation to only a few select bones, rather than to the whole skeleton. Lumberyard has a maximum of 16 virtual layers available for use. Layer 0 is the primary base layer and contains the base full-body animations, joints, and blend spaces. Higher levels contain additive partial-body animations and overwrite animations, meaning that animations in higher layers overwrite animations in lower layers. As long as they don't share the same joints, these animations won't interfere. You can combine all layers into a single layer, which applies them to a character simultaneously.

If an animation played in layer 0 has no controller for a specific bone, the default transformation from the character rig is used instead. Layer 0 is the only layer that supports the root bone and the locomotion locator.

Each layer can play and blend animations and has its own transition queue that handles the blending in and out of animations in the layer. The default behavior for animations in a layer is as follows:
1. Play animation once; then blend it out (weights decrease to 0).
2. Remove animation from the queue when the weight reaches 0.
3. Blend in the next animation (weight increases from 0 to 1).

In Geppetto, only one animation layer is active by default for previewing animations. Any time you select an animation, it plays on the default base layer. You can find the **Animation Layers** listed in the **Scene Parameters** panel of Geppetto.

**To add animation layers using Geppetto**

1. In Lumberyard Editor, click **Tools, Geppetto**.
2. In **Geppetto**, click the drop-down menu next to **Animation Layers** and click on **Add**. The newly added layer becomes the active layer for you to select a new animation from the **Animations** list in the **Asset** panel to assign to the new layer.
3. Repeat this step for as many animation layers as you need. At any point, you can click on a specific animation layer to make it active in order to change the animation playing on that layer.
4. Adjust the blend weight (0 to 1) for each layer.
5. Enable and disable layers using the check boxes next to each layer.

You can enable on-screen debug information to see which animations are queued and playing, as well as information about the applied pose modifiers and IK.

**Accessing Animation Layers using Code**

To access animation layers via code, use the **ISkeletonAnim** object. In the example below, a looping animation starts on layer 2 and is fully blended in 0.5 seconds.

```cpp
ISkeletonAnim& skeletonAnim = ...;
CryCharAnimationParams params;
params.m_nLayerID = 2;
params.m_nFlags |= CA_LOOP_ANIMATION;
params.m_fTransTime = 0.5f;

// Starting the animation by id. Alternatively use StartAnimation to start an animation by name.
skeletonAnim.StartAnimationById(animationId, params);
```

To smoothly blend out animations in a layer, use the **StopAnimationInLayer** function:

```cpp
ISkeletonAnim& skeletonAnim = ...; // Blend out all animations in layer 2 in 0.5 seconds:
skeletonAnim.StopAnimationInLayer(2, 0.5f);
```

To force the transition queue in a specific layer to immediately clear all animations:

```cpp
ISkeletonAnim& skeletonAnim = ...

skeletonAnim.ClearFIFOLayer(layerId);
```

To force transition queues in all layers to clear immediately, use **StopAnimationsAllLayers**, as follows:

```cpp
ISkeletonAnim& skeletonAnim = ...

skeletonAnim.StopAnimationsAllLayers();
```
Working with Blend Shapes (Morphs)

Animated blend shapes, also known as morph target animation, is a method that stores a deformed version of a mesh as a series of vertex positions. In each keyframe of an animation, the vertices are then interpolated between these stored positions.

Blendshape animations are created by adding bones to the base skeleton. This involves explicit name matching between bone names and the blend shape controls.

For blend shape export requirements, see Exporting Blendshapes (p. 17).

Blend Shape Authoring Requirements in Maya

As blend shapes only work for .skin attachments, use Maya to create a base .chr like a cube or triangle that is skinned to the export skeleton.

See the following requirements and guidelines when creating a blend shape scene in Maya:

- All blend shape meshes must exist be in the same world space location as the skinned base mesh. Move your blend shape meshes on top of the skinned base mesh.
- **Smooth bind** at least one joint to the blend shape base mesh.
- Make sure the root joint of the skeleton hierarchy has no (zero) rotations.
- Create an empty SceneRoot group node and a root joint as the top-level node of the deforming skeleton. Do not skin the root joint into your character mesh.
- Set the SceneRoot group node and the root joint both looking forward with their z-axes aligned to the world y-axis and their y-axes aligned to the world z-axis.
- For each blend shape mesh, create a joint in the origin and name it `blend_shape_mesh_name_blendWeightVertex`
- The `_blendWeightVertex` joints should be parented under the root joint for the skeleton hierarchy.
- Manually create the blendWeightVertex joints and connections. Connect and map the weight output range (0 to 1) of the blend shape node to (0 to 100) to the `translateX` attribute of these helper joints.
- Nonrigid deformations require real-time tangent updates to get correct shading. Because such tangent updates are expensive, in order to minimize CPU cost, use vertex colors to transfer a blue (0, 0, 255) painted mask in your DCC tool to mark the most important facial parts.
- Tangent updates only work with 8-weight CPU skinning. To implement that, open the .cdf file and add `flags=8` on the line that lists the applicable skin attachment. This skinning makes the morphs expensive to use, so use it sparingly.

Blend Shape Setup in Lumberyard

Use the following procedure when setting up a blend shape in Lumberyard using Geppetto:

**To set up a blend shape in Geppetto**

1. Create a `SkeletonList.xml` file and place it in the `\Animations` directory and add the following skeleton element block to the file:

   ```xml
   <SkeletonList>
     <Skeleton name="base_skel" file="exported_character_path_filename.chr"/>
   </SkeletonList>
   ```

2. Add a .skin attachment to the skeleton .chr file.
3. By default, Geppetto will add a Joint Attachment. Change this to a Skin Attachment and browse for the .chr file you exported earlier.
4. Enable **Software Skinning** for the blend shape to work.
5. Create a `.chrparams` file.
6. Add an **Animation Set Filter** and point it at the directory of the exported animation file.
7. Browse the directory containing the exported animation file and select the `default` animation.
8. Add a new `.animsettings` file and save it.

**Working with Blend Spaces (Bspaces)**

Lumberyard supports blend spaces, also known as locomotion groups or LMGs, which are related motion parameters that you use to create motion clips. Specifically, an asset's kinematic, physical, and other high-level motion-related parameters are mapped onto corresponding features that are stored in the animation clips. By storing such motion as parameters, you can create controllable interactive animations.

With blend spaces, animation blending is treated as geometry. The structure of a blend space is similar to a character mesh with a vertex buffer and index buffer. Each animation clip represents a point on a coordinate system. Specifically, each animation is associated with a 1D, 2D, or 3D location in a blend space. You can play blend spaces on any layer, and they can contain additive or partial body animation.

Blend spaces (.bspace file format) in Lumberyard are XML-based file maps of animation blends that the Mannequin system uses. A `.comb` file represents a multidimensional blend space. Geppetto supports hot-loading of these XML files. This means you can change the XML file with a text editor, and Lumberyard updates it automatically and renders the result. This makes it ideal for prototyping and experimentation. Almost all parameters are identical for 1D, 2D, and 3D blend spaces.

Lumberyard supports blend space control of the following parameters:

- Move Speed
- Turn Speed
- Travel Angle
- Slope
- Turn Angle
- Travel Distance
- Blend Weight

**Displaying Blend Spaces**

The best way to get a feeling how blend spaces work internally, is to start a simple 2D-BSpaces, visualize it in Geppetto and a play around with the different debug options.

**To display blend spaces**

1. Open Geppetto and load a character that has a blend space file.
2. Click View, Blend Space Preview. This displays the Blend Space Preview window on the right side of the Geppetto window between the Scene Parameters and Properties panels.
3. Detach the Blend Space Preview window from the Geppetto window by clicking the Toggle Floating button. Once detached, adjust the size of the Blend Space Preview window by grabbing it's corners.
4. Under the Assets Panel, under Animations, select the blend space file. The character displays in the preview window at each point on the grid that represents the blend space. The character in the Geppetto window is also animated based on the blend space controls.
5. Use the same viewport controls to navigate within the **Blend Space Preview** window as you would in the Geppetto window.

6. To adjust what part of the blend space is being displayed in Geppetto window, go to the Scene Parameters panel and expand the blend space animation layer to use the sliders to change the blend space's dimensions, such as travel speed and angle.

7. Adjust the blend space dimensions, examples, and annotations listed under the **Properties** panel as needed.

### 1D Blend Spaces

For 1D blend spaces, you can control a single character parameter X, such as movement speed.

In 1D blend spaces all points are on line segments. It is important that p0 points to the lower parameter and p1 points to the higher parameter. If the order is reversed or both parameters are identical, an error results.

Make sure that the line has no uncovered gaps and no overlapping line segments. At runtime, Lumberyard checks whether the input parameter is inside a line segment and then interpolates the value between the two animations.

### 2D Blend Spaces

2D blend spaces involve changing two parameters, X and Y, independently. This means when one parameter is changed, the other parameter stays constant and vice versa. An example of a 2D blend space is a character that moves at different speeds while also turning while moving. When the speed is changed, the turn radius (body angle) stays the same; and when the turn radius is changed, the speed is not affected.

In 2D blend spaces all points are on planar triangles and quads. Looking down onto the blend space, annotations occur counterclockwise for triangles and quads.

Make sure that the plane has no overlapping triangles and quads and no gaps or holes that are not covered with a face. At runtime, Lumberyard checks whether the input parameters fall inside a plane, and then interpolates the values between the three animations (for triangles) or four animations (for quads).

### 3D Blend Spaces

For 3D blend spaces, three separate parameters X, Y, and Z are changed independently. For example, character speed, turn radius, and travel angle can be changed.

In 3D blend spaces all points are inside of volume tetrahedrons, pyramids, and prisms. All have a ground plane (3 or 4 points) and a tip (1 or 2 points). If the tip points up, the vertices on the ground plane must be annotated counterclockwise. If the tip points down, the vertices are annotated clockwise.

Make sure that the space has no overlapping volumes and no holes that are not covered with a volume. At runtime, Lumberyard checks whether the input parameters are inside of one those volumes and then interpolates the values between those animations.

3D blend spaces are more difficult to debug, even with a very structured design. Fortunately, many higher dimensional blend spaces are a combination of simple lower dimensional blend spaces. This relationship makes it possible to combine two 2D blend spaces into a 3D space and two 3D blend spaces into a 4D blend space.

### Number of Assets for Movement

Four assets are the minimum, but eight are the recommended minimum for realistic 360-degree movement. Diagonal blends usually don't look as good as forward and sideways motions.
diagonal blends can create foot-sliding, foot-crossing, and foot dipping through the ground if you only use four.

Another issue is hip rotation. Usually the hips point to the right when sidling right and to the left when sidling left. However, doing quick left to right side steps looks like Samba dancing. For best results, keep hip orientation static in each blend space, create a new blend space for each hip rotation, and play an explicit transition to adjust the gait. In this situation, 16 assets may be needed.

Debug Information

The following information is provided in the Blend Space Preview window:

• All animation files available in the blend space, as a mini version of the character model. You can control the size of the model using the slider near the top right of the preview window.
• Each model has either a white or red spinning cube at its root joint.
• Each cube has an index. This is the order that the animation clips appear in the .bspace XML file, including all pseudo examples.
• A green wireframe quad shows which assets are currently considered in the blend. In a 2D blend space there are either triangles (blends between 3 assets) or quads (blends between 4 assets).
• A red flashing cursor appears inside a triangle or quad. You can control the cursor with the blend space sliders and see which assets contribute to the final blend.
• The current dimension values correspond with the current slider values set by the Scene Parameters animation layer.

Animation Events

Using Geppetto, you can add character animation events by double-clicking in the Playback timeline window. If you can right-click in the timeline, you can jump to previous and next events. Each animation can have multiple events specified.

If you need to create a large number of animation events, click View and select Animation Event Presets. This creates a new Animation Event Presets panel above the Properties panel.

This provided you a set of quickly-accessible animation events, which you can add to the playback timeline with a double-click. Keys with events corresponding to the presets are colored the same in the timeline.

Animation events are also accessible from the Properties pane for an animation. These are stored in an .animevents file, which is referenced from the .chrparams file, which contains lists of animation events per animation.

You will need to create an .animevents file per character skeleton unless the character shares skeletons and animations.

To create the .animevents file

2. In the .xml file, add the following tags: <anim_event_list></anim_event_list>.
3. Name the file and include the .animevents extension. Save the file in the same directory as the animations to which the file will apply.

To update the .chrparams file

1. In Geppetto, in the Assets panel, expand Skeletons, and navigate to and select the character's skeleton (*.chrparams) file.
2. In the Properties panel, locate the Events field.
3. Click the folder icon next to the empty property field and assign a directory where the .animevents file will be located for this character's skeleton.

**Note**
You can assign only one .animevents file per .chrparams file.
4. After assigning the .animevents file, click Save to save your changes to the .chrparams file.

**Locomotion Locator Animation Best Practices**

The locomotion locator, or Locator_Locomotion bone, is a node that is required for nonlinear or nonuniform character motions, such as a start or stop transition that has peaks and troughs in acceleration. For best results, consider doing the following:

- This bone must have the same orientation as the root joint and the SceneRoot node, which is the positive y-axis in the local coordinate system. Otherwise animations are rotated to match the orientation of the locomotion locator bone. This only affects the animation and not the skeletal orientation.
- The first and last keyframe of your animation cycle should match. The locomotion locator position relative to the character on the first keyframe should also match the position relative to the character on the last keyframe.
- The orientation of the locator in an idle-to-move transition should remain looking forward until keyframe 10.
- Make sure that orientation changes (left, right, left reverse, or right reverse) occur in the following 6 frames so the new orientation is complete at keyframe 16.
- When changing the orientation 180 degrees for reverse transitions, make sure you rotate the locator 0.1 degrees back to its original orientation to avoid flipping the character.
- For swimming transitions or vehicle transitions, the locator can be a straight blend between the ground position of 0,0,z and end at the Bip01 location and forward-looking direction (positive y-axis) of the character.
- For animation loops, set keys for the start and end of the animation only if you need to add a locator to them. They are technically not needed but can be useful for batch processing.

**Streaming Character Animations**

Animation can be a very memory-intensive resource. Limited memory budgets, a high number of animated joints and high animation quality requirements makes it undesirable to have all animations loaded in memory all the time. Lumberyard alleviates this issue by streaming asset files in and out as needed.

Animation data is divided into header data and controller data. Given the extreme size difference between controller and header data, only controller data is streamed in and out. The header data for all animations is kept in memory at all times.

**Animation Header Data**

The header contains generic information for an animation such as filename, duration, and flags. Header data is stored in .CAF files and in the animations.img file.

CAF files contain header information for a single animation, while the Animations.img file contains header information for all animations in the build. The Animations.img file is obtained as a result of processing all the animations using the Resource Compiler.
Animation Controller Data

The controller contains animation curves for each joint's position and orientation values needed to play the animation. Even when compressed, controller data can easily take up more than 95% of the total memory required for an animation.

The controller data for animations is stored in CAF files, which contains controller information for a single animation, and a DBA file, which contains controller information for a group of animations.

Mannequin System

Mannequin builds on top of the Geppetto tool to make it easier to construct complex, interactive character animations. Mannequin provides animation layering, blending, additive animations, and partial body animations.

The core of Mannequin is the ability to define families of movements that are variations on a theme (e.g. running injured, running exhausted, running slow, etc.), and to author smooth transitions between those families. Each variation in a family is called a fragment. Fragments are grouped together into families by sharing a fragment ID. Each fragment can carry one or more tags (e.g. tired, injured, gun-in-hand) that selects fragments from within a family during playback, allowing easy authoring of highly varied and situation-specific animation sequences.

With Mannequin you can simplify complex animation code and avoid manually constructing this degree of realism. You can also author preview sequences using your fragments and transitions, reducing iteration time and allowing you to retest scenarios as your animation setup evolves. The Mannequin runtime allows you to play sequences of fragments that smoothly transition from one to the other under the control of C++ code or the flow graph visual scripting system.

Topics

- Mannequin System Files (p. 82)
- Creating a Mannequin Entity (p. 85)
- Using Mannequin Editor (p. 86)
- Using the New Mannequin Setup Window (p. 111)
- Synchronizing Multiple Characters (p. 115)
- Using Flow Graph with Mannequin (p. 116)
- Debugging Mannequin System Issues (p. 116)

Mannequin System Files

The Mannequin system uses a variety of file types.
With the exception of the *.Sequence.xml file, you must use a text editor to manually create all other .xml files. You must save these files in the Animations\Mannequin directory, and you can optionally create subfolders by character. See below for example files.

**Controller Definition File (*.ControllerDefs.xml)**

Used by the game and by Mannequin Editor to define a mannequin setup. This file is typically referred to from the character Lua file and Mannequin Preview file.

```xml
<ControllerDef>
  <Tags filename="Animations/Mannequin/Sample/Character_Tags.xml"/>
  <Fragments filename="Animations/Mannequin/Sample/Character_FragmentIDs.xml"/>
  <FragmentDefs>
  </FragmentDefs>
  <ScopeContextDefs>
    <Char3P />
  </ScopeContextDefs>
</ControllerDef>
```
Animation Database File (*.adb)

Used by the game and by Mannequin Editor to store fragments and transitions. This is typically referred to from the character Lua file and other systems such as the hit death reaction system.

Tag Definition File (*.Tags.xml)

Used by the game and by Mannequin Editor to store tag definitions. The controller definition and animation database files refer to this file.

FragmentID Definition File (*.FragmentIDs.xml)

Used by the game and by Mannequin Editor to store FragmentID definitions. The controller definition and animation database files refer to this file.

Character Definition File (*.cdf)

Used by the game and by Mannequin Editor to store the main character (.chr) as well as any attachment definitions.

Preview Setup File (*.Preview.xml)

Used by Mannequin Editor to determine which controller definition file, animation database file, and character to load.

Sequence File (*.Sequence.xml)

Used by Mannequin Editor to store animation sequences.

Setting up Mannequin files

You must manually set up and edit some Mannequin files. Once these files are set up, you can use the Mannequin Editor to verify that the character displays in the viewport. To do so, click File, Load Preview Setup, and select the *Preview.xml file. You must select File, Save Changes any time a change is made.
When setting up the *ControllerDefs.xml file, the file name should match the character name for easier recognition. You should also reference this name in the *Preview.xml file.

**To set up the *ControllerDefs.xml file**

1. Set the Tags filename path to point to the Tags.xml file.
2. Set the Fragments filename path to point to the FragmentIDs.xml file.
3. Save the file.

**Setting up the *Preview.xml file**

This file name should match the name of the character so it's easier to recognize.

**To setup the *Preview.xml file**

1. Open the Character_Preview.xml file in a text editor.
2. Set the controllerDef filename path to point to the appropriate Controller Definition file.
3. Set the contextData model path to point to the character model .cdf file you want to use in Mannequin.
4. Save the file.

**Setting up the .adb file**

You will also need to set up the Animation Database (.adb) file and assign it to your *Preview.xml file. Once the .adb file is assigned to the *Preview.xml file, Mannequin fragments can be added.

**To setup the .adb file**

1. In Mannequin Editor, choose File, Context Editor.
2. Select the MainCharacter entry.
3. Click the Edit button.
4. For the Database field <no animation database (adb)> entry, click the + (Add) button.
5. In the Edit Context window, enter the name of the .adb file. Click OK when done.
6. Verify the Database field in the Edit Context window points to the .adb file.
7. Click OK in the Edit Context window.
8. Click OK in the Context Editor window.

**Creating a Mannequin Entity**

You can use the Mannequin system to control complex characters, which are often created by code, and you can use the Mannequin object entity type to create a character that can host a Mannequin setup and support any feature of that system.

**To create a Mannequin entity**

1. In Lumberyard Editor, in the Rollup Bar, click Entity and then select Anim\MannequinObject.
2. Drag the Mannequin object to the viewport.
3. In the Entity Properties, click each of the following to assign the specific files:
   - **ActionController** – Select a *ControllersDef.xml file.
• AnimDatabase3P – Select an *.adb database file.
• Model – Select a character *.cdf file.

Using Mannequin Editor

Mannequin Editor is the primary tool for creating and managing complex character animations.

To open Mannequin Editor

• In Lumberyard Editor, click Tools, Animation, Mannequin Editor. You can also open Mannequin Editor from its icon on the main toolbar for Lumberyard Editor.

Fragments Browser

The Fragments browser occupies the left pane of Mannequin Editor by default, and contains the FragmentID Editor tab. The Fragments Browser lists all fragments stored in the animation .adb database file. You use the Fragments browser tab in conjunction with the Fragment Editor tab to create fragments, change fragment tags, and create FragmentIDs.

To access the Fragments Browser, click the Fragments tab at the bottom left of Mannequin Editor.

You use the FragmentID Editor to edit FragmentID names and fragment definition properties that are stored in the controller definition ControllerDefs.xml file.

Fragment Editor

The Fragment Editor occupies the central pane of Mannequin Editor. You use the Fragment Editor to edit mannequin fragments and animation clip properties.

To access the Fragment Editor, click the Fragment Editor tab at the bottom of Mannequin Editor. You can also start the editor by double-clicking a fragment in the Fragments browser.

Tag Definition Editor

You use the Tag Definition Editor to create and edit tag definition files (*.Tags.xml), which are used for labeling fragments and transitions. To open the Tag Definition Editor, choose File, Tag Definition Editor. You can also access it by clicking on the Tag Definition Editor button in the FragmentID Editor.

Transitions Browser

The Transitions browser occupies the left pane of Mannequin Editor. The Transitions browser lists all transitions stored in the animation .adb database file. You use the Transitions browser in conjunction with the Transition Editor to create transitions.

To access the Transition Editor, click the Transitions tab at the bottom left of Mannequin Editor.

Transition Editor

The Transition Editor occupies the central pane of Mannequin Editor. You use it to edit and display mannequin transitions.

To access the Transition Editor, click the Transition Editor tab at the bottom of Mannequin Editor. You can also start it access it by double-clicking a transition in the Transitions browser.
Sequences Browser

The Sequences browser occupies the left pane of Mannequin Editor. The Sequences browser lists all the XML sequence files that are stored in the default sequences directory. You use the Sequences browser to select the sequences that you want to open in the Sequence Previewer.

To open the Sequences browser, click the Sequences tab at the bottom left of Mannequin Editor.

Sequence Previewer

The Sequence Previewer occupies the central pane of Mannequin Editor. You use the Sequence Previewer to edit and view mannequin sequences from an XML sequence file or to test a new sequence of fragments before saving it to a file.

To access the Sequence Previewer, click the Previewer tab at the bottom of Mannequin Editor.

Animation Database Editor

You use the Animation Database Editor to create .adb files and to edit the rules that determine which fragments are stored in a specified .adb file.

To open the Animation Database Editor, choose File, Animation Database Editor.

Context Editor

You use the Context Editor to edit the preview setup (*Preview.xml) file. Mannequin Editor needs the preview setup file to determine which controller definition (*ControllerDefs.xml) file to load, which animation database (.adb) file to use, and which characters to use in specific scope contexts.

To open the Context Editor, choose File, Context Editor.

Mannequin Error Report

The Mannequin Error Report displays the validation results for any files that are opened in Mannequin. Validation is performed every time you open a new Mannequin-related file, with errors and warnings listed for each fragment. You can copy validation results to the clipboard, email, or open in Microsoft Excel.

To see the Mannequin Error Report, click the Mannequin Error Report tab at the bottom of Mannequin Editor.

Mannequin Fragments (Clips)

The fragment is the basic building block within the Mannequin system. A fragment is a layered collection of time-sequenced animation clips and procedural clips such as poses, attachments, and sounds. You can transition from one clip to another, speed up clips, loop them, or cut them up. This is similar to other nonlinear animation tools. Instead of starting a specific animation directly, the fragment containing the animation is called first. Fragments are defined by their FragmentIDs and tags.

FragmentIDs represent an animation state, such as crouching, idling, or aiming. You use the FragmentID to request fragments. Note that multiple fragments often share the same FragmentID.

You use tags to label fragments with easy-to-remember names, such as blink, yawn, or step. If multiple fragments share the same FragmentID and tag, the fragments are designated as options.
Animators create the animation clips and fragments, while game developers define the FragmentIDs and tags used in Mannequin.

**Topics**
- Managing Mannequin Fragments (p. 88)
- Fragment Selection Process (p. 88)
- Using Animation Clips in Fragments (p. 89)
- Using Procedural Clips in Fragments (p. 89)
- Adding Layers to a Fragment (p. 101)
- Managing Fragment Preview Sequences (p. 102)

**Managing Mannequin Fragments**

Use the Mannequin Editor to create, copy, and delete fragments.

**To create, copy, or delete a fragment**

Open Mannequin Editor (p. 86), choose the **Fragments** tab at the bottom, and do the following. The Fragments browser (panel) is displayed on the left.

- To create a fragment, select the applicable FragmentID, and then click **New**.
  
  **Tip**
  Alternatively, you can also drag the corresponding animation from within Geppetto and drop it onto the FragmentID.

- To copy a fragment, drag it to the desired location.

- To delete a fragment, select it and then choose **Delete**.

**Fragment Selection Process**

The following process determines which fragment gets selected for use when a game request is made.

- Determine FragmentID for fragment
- Determine scope mask for FragmentID
- Determine scopes assigned to FragmentID
- Determine scope context assigned for each scope
- Determine scope tags assigned for each scope
- Find best matching fragments in the animation database .ADB file assigned to the scope context for each scope. A matching fragment must contain all the scope tags for a scope.
- Ranking matching fragments using tag priorities. Fragments are displayed in the Fragments panel according to rank.
- If there are multiple options with the same tags, the option index is used to select the fragment.

First, the Mannequin system determines which scopes are assigned to the requested fragmentID by looking up the scope mask for the fragmentID. Typically the fragmentID determines the scope mask by itself, but it is possible to specify 'overrides' and select different scope masks based on the global tagstate and requested frag tags. See the file format section in the article on the controller definition file for more on how this is set up. Also, if the calling action requests a specific SubContext, the scope mask and global tags coming from this SubContext's definition extends the ones from the original request. Finally, the scope mask can optionally be extended by the action's 'forced scope mask'.

---

88
Using Animation Clips in Fragments

You can easily add animation clips to a fragment and move them around the fragment timeline as desired.

To add an animation clip to a fragment

1. In the Mannequin Editor, from the Fragment Editor pane, select the applicable fragment or create a new fragment.
2. Add an animation layer to the fragment by right-clicking the scope for the fragment in the Fragment Editor, going under Add Layer, and clicking AnimLayer. It is not possible to add animation clips until there is an animation layer available.
3. In Geppetto, select the animation from the Animation list, and then drag the animation to the desired location in the timeline window for the fragment.
4. Add an empty animation clip by double-clicking on the timeline. With the empty clip selected, you can assign an animation by clicking on the folder icon for the Animation property under Anim Clip Properties.

Understanding Fragment Clip Zones

The timeline window contains various locations and zones. Understanding them and their effect on fragments can help you add animation clips to a fragment.

The timeline shows various aspects of a clip:

- Blend-in period of the first clip.
- The period where the first clip is playing.
- After the first clip has finished, and last key is repeated by default.
- Blend-in period of the second clip.
- The period where the second clip is playing.

Normally, the second clip is positioned toward the end of the first clip so there aren't any repeating frames. You can also increase or decrease the blend-in time by dragging the vertical bars.

Moving and Snapping Animation Clips

You can drag a clip to move it along the fragment timeline. The default dragging behavior is to snap to the beginning, end, or blend time of a clip.

To snap the clip to the timeline, begin dragging the clip and then press Shift as you continue to drag. This snaps the clip to the timeline markers and ignores the other animation clips.

To disable snapping, begin dragging and then press Ctrl as you continue to drag. You can now drag the clip to any point on the timeline without snapping to the other clips or to the timeline.

Using Procedural Clips in Fragments

Procedural clips are code snippets that you insert into fragments and run alongside animation clips in that fragment. Like animation clips, procedural clips can be started, stopped, and blended. They are grouped into Lumberyard (CryAction) and game (GameDLL) types.

Procedural clips can range from playing sounds, controlling joints on a character, or aligning an entity to a location. Procedural clips communicate with game code by means of parameters or procedural contexts.
To edit procedural animation clips, you use the Fragment Editor within Mannequin Editor.

To add a procedural clip to a fragment

1. In Mannequin Editor, from the Fragments Browser, select the applicable fragment or create a new fragment.
2. Add a procedural layer to the fragment by right-clicking on the scope for the fragment in the Fragment Editor, going under Add Layer, and clicking ProcLayer. It is not possible to add procedural clips until there is a procedural layer available.
3. Double-click in the timeline on the ProcLayer to add a new empty procedural clip.
4. Set the procedural clip Type property in the Procedural Clip Properties pane.

CryAction Procedural Clips

The following are classified as CryAction procedural clips.

**ActionEvent clip**

Sends a CryMannequin event to the action controlling this fragment. Specifically, calls IAction::OnActionEvent.

**Event Name**

The name of the event to send.

**AimPose clip**

Low-level clip to start an AimPose asset. Uses the AimTarget parameter as the target, if it exists. If not specified, the target is 10m in front of the entity.

**Animation**

The Aimpose asset.

**Blend**

The fade-in duration.

**Blend Time**

The smoothing time for the spherical coordinates. Higher numbers mean the longitude and latitude have faster smooth aiming at the target.

**Animation Layer**

The layer (0–16) to play the Aimpose on.

**Note**

This works differently than the layer parameter inside the LookPose procedural clip, which is a layer index relative to the scope's first animation layer. For more information on scopes, see Mannequin Scopes (p. 103).

**AI Signal clip**

Sends an AI signal directly to the AI actor interface of the entity on which the clip is playing.

**EnterAndExitSignalNames**

Signal names sent on the start and finish of the clip, separated by a | character.
AttachEntity clip
Attaches an entity to a specific attachment point, and then detaches it on exit.

Attachment Name
The name of the attachment point.

Param Name with EntityId
The name of the parameter that stores the EntityID of the entity to attach.

AttachProp clip
Attaches a .chr, .skel, or .cga to a specific attachment point (and detaches on exit).

Object Filename
Name of the .chr, .skel, or .cga to attach.

Attachment Name
The name of the attachment point.

Audio clip
Runs the audio translation layer (ATL) triggers.

Start Trigger
(Optional) ATL trigger to execute at the start.

Stop Trigger
(Optional) ATL trigger to execute at the end.

Attachment Joint
(Optional) name of a joint on which to execute the trigger.

Play Facial
Requests facial animation to match the sound.

Sound Flags
(Reserved)

FlowGraphEvent clip
Sends events to the flow node Actor ProcClipEventListener.

Enter Event Name
Name of the event to send at start.

Exit Event Name
Name of the event to send at end.

HideAttachment clip
Hides an attachment for the duration of the clip.
Attachment Name

Name of the character attachment to hide.

IKLayerWeight clip
Controls the weight of an animation layer by a joint's X value.

Joint Name
The joint whose X value controls the layer weight.

Scope Layer
The index of the layer within this scope that this clip should control.

Invert
Use (1.0 - \texttt{value}) as the weight.

LayerAnimSpeed clip
Controls the speed of an animation that is playing in the same scope as this procedural clip through code. The Blend value is not used.

LayerAnimSpeedParam
The name of the floating point parameter that stores the speed value (0 by default).

ScopeLayer
The layer index within the scope of the animation that you want to control.

Invert
Uses (1.0 – \texttt{value}) as the speed.

LayerManualUpdate clip
Controls the (normalized) time of an animation that is playing in the same scope as this procedural clip through code.

Param Name
The name of the floating point parameter that stores the normalized time value (0 by default).

Scope Layer
The layer index within the scope of the animation that you want to control.

Invert
Uses (1.0 – \texttt{value}) as the normalized time.

LayerWeight clip
Controls the weight of an animation layer through code.

Layer Weight Param
The name of the floating point parameter that stores the weight to apply to the layer
Scope Layer

The layer index within the scope of the layer that you want to control.

Invert

Uses \((1.0 - \text{value})\) as the normalized time.

LookPose clip

Low-level clip to start an LookPose asset. Uses the \(\text{LookTarget}\) parameter as the target, if it exists. If not specified, the target is 10m in front of the entity.

Animation

The Lookpose asset.

Blend

The fade-in duration.

Blend Time

The smoothing time for the spherical coordinates. Higher numbers mean the longitude and latitude have faster smooth movement toward the target.

Scope Layer

The layer to play the Lookpose on, relative to the scope's first animation layer.

Note

This works differently than the layer parameter inside the AimPose procedural clip, which is the actual layer number (0–16).

ManualUpdateList clip

Controls the normalized time of animations playing in multiple layers through code.

Param Name

The name of the parameter of type \(\text{SWeightData}\) (four floating-point weights), where the parameter stores the segment normalized time values for the layers.

Scope Layer

The layer index within the scope of the first layer that contains animation that you want to control. All layers after that within this scope are also controlled (up to four layers).

Invert

Use \((1.0 - \text{value})\) as the weight.

ParticleEffect clip

Plays a particle effect.

Effect Name

Name of the particle effect to spawn.

Joint Name

Optional joint to attach the emitter to.
**Attachment Name**

Optional attachment interface name to attach the emitter to.

**Position Offset, Rotation Offset**

Local-space offset of the emitter. If Joint Name or Attachment Name is given, the offset is relative to the host entity.

**Clone Attachment**

If Attachment Name is given, create a copy of the given interface instead of using it directly. This allows for more than one effect to play on the same attachment. Disabled by default.

**Kill on Exit**

Explicitly remove all spawned particles instead of letting them die out on their own. Disabled by default.

**Keep Emitter Active**

Keep emitter alive after the procedural clip has ended. Disabled by default.

**Note**

Use with care - if the particle effect goes away on its own, there is no other way to get rid of the effect after it started.

**PositionAdjust clip**

Procedurally moves the entity towards a target position over time. The target position is taken from the TargetPos parameter, which must be set for the clip to play. Used to align characters to ledges.

**Blend**

Duration of the adjustment.

**Offset, Yaw**

Additional offset on top of the target position.

**Ignore Rotation**

Checks to ignore rotation.

**Ignore Position**

Checks to ignore position.

**PositionAdjustAnimPos clip**

Moves the entity from the source position (its origin in the DCC tool) of the animation to the target position. If the animation contains movement, this clip might not behave as expected as the delta is only calculated at the start of the animation. In this case, use the PositionAdjustAnimPosContinuously clip instead. The target position is taken from the Param Name parameter.

**Blend**

Duration of the adjustment.

**Param Name**

(Optional) Name of the parameter to use. If not specified, uses the TargetPos parameter.
Ignore Rotation
Check to ignore rotation.

Ignore Position
Check to ignore position.

**PositionAdjustAnimPosContinuously clip**
Moves the entity from the source position (its origin in the DCC tool) of the animation to the target position. The target position is taken from the `TargetPos` parameter, which must be set for the clip to play.

**Blend**
Duration of the adjustment.

**PositionAdjustTargetLocator clip**
Takes the character assigned to the specified scope, typically a dependent scope, and moves the entity towards the location of a specific joint of this character.

**Blend**
Duration of the adjustment.

**Target Joint Name**
Name of the joint to align to.

**Target Scope Name**
The scope that has the dependent character attached that you want to align to.

**Target State Name**
Not used.

**SetParam clip**
Sets a float parameter to a certain value.

**Param Name**
The name of the parameter.

**Blend**
The time it takes to reach the target value.

**Target**
The target value.

**Exit Target**
The value to go to after the clip ends.

**WeightedList clip**
Controls the weight of consecutive layers through code.
Param Name

The name of the parameter of type SWeightData (four floating-point weights), which stores the weights for the layers.

Scope Layer

The layer index within the scope of the first layer that you want to control. All layers after that within this scope are also controlled (up to four layers).

Invert

Uses \(1.0 - \text{value}\) as the speed.

Game Procedural Clips

The following are classified as GameDLL procedural clips.

Aiming clip

Requests that the Aimpose be enabled.

Blend

Fade-in duration for the Aimpose.

AimSmoothing clip

Relies on Aimpose or Aiming scope setup. Controls smoothing parameters for the polar coordinates while moving toward or following a target.

Smooth Time Seconds

The "smoothing time" for the spherical coordinates. Higher numbers mean the longitude or latitude have faster smooth movement towards the target.

Max Yaw Degrees Per Second

Maximum degrees per second in the yaw direction.

Max Pitch Degrees Per Second

Maximum degrees per second in the pitch direction.

AttachPnt clip

Attaches the pick-and-throw weapon.

Attachment Point

Name of the attachment interface to use.

ColliderMode clip

Overrides the ColliderMode for the character.

Valid values:

- Undefined (give up control)
• Disabled (no collisions)
• GroundedOnly
• Pushable
• NonPushable
• PushesPlayersOnly
• Spectator

CompromiseCover clip
Tells the AI system that cover has been compromised.

CopyNormalizedTime clip
Synchronizes animation within two layers by automatically copying over the segment normalized time from an animation in one layer to an animation in another layer.

Source Scope
The scope from which to copy.

Source Layer
The layer within the source scope to look for the source animation.

Layer
The layer within the current scope that contains the animation that you want to synchronize

FacialSequence clip
Plays a facial sequence.

Filename
The facial animation sequence .fsq file to play

Continue After Exit
Whether to continue playing the sequence after the clip ends. Ignored when looping the sequence, in which case the default behavior is used, so the sequence stops playing when the clip ends.

Looping
Whether to loop the sequence.

Looking clip
Relies on Lookpose or Looking scope setup. Requests the Lookpose to be enabled. Blend-in time is used as fade-in time for the Lookpose.

Blend
Fade-in duration for the Lookpose.

MovementControlMethod clip
Override the movement control method of the character.
Horizontal

Horizontal movement control method. Valid values:

- 0: Undefined (no override)
- 1: Entity driven
- 2: Animation-driven
- 3: Animation-driven with collision in the horizontal plane

Vertical

Vertical movement control method. Valid values:

- 0: Undefined (no override)
- 1: Entity-driven
- 2: Animation-driven

Ragdoll clip

Makes a character turn into a ragdoll and optionally blend back to animation.

Blend

Defines the time range during which the character starts randomizing.

Sleep

When set to 0, the AI exhibits ragdoll behavior. When set to 1, the AI stays alive during the ragdoll phase and blends back to animation.

Stiffness

Determines how much the ragdoll behavior follows the animation.

Note

The Sleep parameter is only used by the blend-from-ragdoll game code, which is triggered by calling CActor::Fall(). This triggers the CAnimActionBlendFromRagdollSleep, which makes the character exhibit ragdoll behavior: It plays the fragment with fragmentID BlendRagdoll and tags containing standup+blendin+ragdoll. This fragment has to contain a Ragdoll clip with the sleep value set to 1.

For standing up, a CAnimActionBlendFromRagdoll is started after the ragdoll phase has ended. This action relies on all possible standup animations to be an option for the fragmentID BlendRagdoll and tags containing standup+blendout. The best matching animation is chosen based upon the first frame of these.

SetStance clip

Tells an AI character it is in a certain stance. It does not trigger stance-change animation. This is useful to annotate an animation that ends up in a stance other than it started in, such as in a scripted sequence that can be interrupted. When the sequence is interrupted, the game knows the AI is in another stance.

Stance

Stance name. Valid values:

- Null
- Stand
Lumberyard Legacy Reference
Using Mannequin Editor

- Crouch
- Prone
- Relaxed
- Stealth
- Alerted
- LowCover
- HighCover
- Swim
- Zero-G

**SwapHand clip**

Temporarily move an attachment from the right hand to the left. This is hardcoded to use the attachment names `weapon` and `left_weapon`.

**TurretAimPose clip**

Controls aiming and aimpose of the turret entity.

**Blend**

The fade in time of the Aimpose.

**Animation**

The Aimpose asset to use.

**Blend Time**

Unused.

**HorizontalAimSmoothTime**

The smoothing time for the yaw direction.

**VerticalAimSmoothTime**

The smoothing time for the pitch direction.

**Max Yaw Degrees Per Second**

Maximum degrees per second that the turret rotates in the yaw direction.

**Max Pitch Degrees Per Second**

Maximum degrees per second that the turret rotates in the pitch direction.

**WeaponBump clip**

First-person weapon bump animation that occurs when the player lands.

**Time**

The amount of time that the bump animation plays.

**Shift**

How much the weapon moves on screen after the player lands.
Rotation

How much the weapon rotates.

WeaponPose clip

Places the weapon on a specific location on the screen. It has three modes: right hand, left hand, and zoom. Only one of these modes can be active at a time; however, more than one clip can run in parallel.

Pose Type

The default is 0, which means right hand. This changes the weapon's position on screen starting from the idle pose position. A value of 1 means zoom, which places the weapon on the screen when the player decides to zoom in. A value of 2 means left hand, which can be used to modify the original base pose to accommodate underbarrel attachments.

Zoom Transition Angle

The default is 0, which defines the angle that the weapon rotates during a zoom transition. Zoom Transition Angle is only read if Pose Type is set to 1 (zoom). Otherwise this parameter is totally ignored.

Position, Rotation

Defines the pose itself as an offset to the base pose. Rotation is defined in angles.

WeaponRecoil clip

Activates the recoil behavior on the weapon. It triggers a recoil animation every time the weapon fires.

Damp Strength

How quickly the weapon comes back to rest pose after a kick.

Fire Recoil Time

Attack time of the recoil kick. A value of 0 applies the kick in a single frame, which is not recommended, since it can make the animation look jerky.

Fire Recoil Strength First, Fire Recoil Strength

The kick strength. Fire Recoil Strength First has the same behavior as Fire Recoil Strength but is applied to the first shot only. For best results in rapid fire modes, make Fire Recoil Strength First much higher than Fire Recoil Strength.

Angle Recoil Strength

The degree of deviation the weapon experiences after each shot.

Randomness

The overall organic feeling of the recoil animation.

WeaponSway clip

This clip activates the laziness effect on the player's moving hands. Careful setup of the clip simulates different weight feelings for different weapons. After the clip is activated, it starts reading the player movement and computes weapon offsets in real time.

Ease Factor Inc, Ease Factor Dec

How much it takes for the look poses to blend in (Inc) or out (Dec) when player looks around
Velocity Interpolation Multiplier

Fine tune control for strafing.

Velocity Low Pass Filter

The filter applied to the player movement to make the sway more reactive or intensive.

Acceleration Smoothing

Helps make strafe poses less linear and more realistic.

Acceleration Front Augmentation

The degree to which it makes more sense for the strafe poses to move back and forth as opposed to left and right.

Vertical Velocity Scale

Changes the look poses behavior when player is going up or down a ramp.

Sprint Camera Animation

Do not use.

Look Offset

The degree to which the weapon moves around the screen while player looks around.

Horiz Look Rot

The rotation applied to the weapon when the player looks left and right.

Vert Look Rot

The rotation applied to the weapon when player looks up and down.

Strafe Offset

The degree to which the weapon moves when player moves around.

Side Strafe Offset

The rotation of the weapon when the player starts strafing either to the left or to the right.

Front Strafe Rot

The rotation of the weapon when the player starts moving forward or backward.

WeaponWiggle clip

Activates weapon wiggling and shaking.

frequency

Shake frequency.

intensity

Shake intensity.

Adding Layers to a Fragment

You can add multiple layers of animation clips to one fragment. In these layers, you can place additive or override animations to add variation to the base layer's animation. In some instances, the number of
layers you can add may be limited by the scope. For information about scope, see Creating and Editing Scopes (p. 103).

**To add a layer to a fragment**

1. In Mannequin Editor, from the Fragment Editor pane, right-click the fragment scope, and then click Add Track, AnimLayer.
2. If you're adding a procedural clip layer instead of an animation layer, when you right-click on the fragment scope, go to Add Track and click on ProcLayer. Currently, when you add a new layer, it is added directly below the lowest layer. You cannot change the order of layers at this time, instead, just reorganize the clips as necessary.

**Managing Fragment Preview Sequences**

You can save, load, and view fragment preview sequences in Mannequin Editor.

**To save a fragment preview sequence**

1. In Mannequin Editor, in the Sequences browser, under Sequences, select the sequence.
2. Click Previewer, Save Sequence. Name the sequence and click Save.

**To load a fragment preview sequence**

1. In Mannequin Editor, in the Sequences browser, under Sequences, select the sequence.
2. Click Previewer, Load Sequence.

You can preview how fragment sequences look without actually running the game. This is useful for debugging sequences and previewing what-if scenarios, such as how the game would look if requesting the Move after Idle while Kneeling fragment sequence, for example.

**To view a fragment preview sequence**

1. In Mannequin Editor, click the Previewer tab at the bottom.
2. Select the sequence and click the start button. You can also rewind and fast forward through the sequence.

**Mannequin Fragment IDs (Animation States)**

A FragmentID is the main label under which a fragment is stored.

FragmentIDs are character animation states, such as moving, idling or firing. Game code uses a FragmentID to access a fragment. Typically, a number of different fragments may be assigned to the same FragmentID. For example, the animation could include several different moving fragments, such as moving while standing, moving while crouching, or moving plus some random variation.

Typically, a game developer creates a different FragmentID for every basic character animation state, while animators create animation clips and the associated fragments for those FragmentIDs.

You can create and edit FragmentIDs in Fragment Editor within Mannequin Editor. You store the FragmentIDs in a FragmentID definition file (*.Actions.xml), which is referred to from the controller definition file (*.ControllerDefs.xml).

If animations are required between FragmentIDs, you can use a transition.
Mannequin Scopes

Typically, individuals portions of a character's body will be in different animation states. Scopes are animation channels assigned to the parts of a character's body on which fragments are triggered and played. For example, one animation fragment can be played for the entire body, another fragment for the lower body, another fragment for the torso, and another fragment for the head. These scoped animations can be played independently or synchronized together.

To create and edit scopes, you modify the following parts of the controller definition file (*ControllerDefs.xml).

- Primary entity (character)
- Attached entities (head, weapon)
- Animation layers
- Animation database for fragments (.adb)

Topics
- Creating and Editing Scopes (p. 103)
- Creating and Editing Scope Contexts (p. 104)
- Using Scope Masks (p. 105)
- Playing Fragments on Scopes (Actions) (p. 105)

Creating and Editing Scopes

Mannequin scopes are stored in the controller definition *ControllerDefs.xml file, which contains the setup of a mannequin character.

The following shows an example *ControllerDefs.xml file. You use FragmentID Editor in Mannequin Editor to edit the scope masks and related flags. To edit the remaining sections, you need a text editor. The FragmentID Editor appears when you create a new FragmentID in the Fragments pane.

```xml
<ControllerDef>
  <Tags filename="Animations/Mannequin/ADB/sampleTags.xml"/>
  <Fragments filename="Animations/Mannequin/ADB/sampleFragmentIds.xml"/>
  <SubContexts/>
  <FragmentDefs>
    <move scopes="FullBody+Torso" flags="Persistent"/>
    <burst_fire scopes="Torso+Weapon">
      <Override tags="heavyMortar" fragTags="boosted" scopes="Torso"/>
    </burst_fire>
  </FragmentDefs>
  <ScopeDefs>
    <FullBody layer="0" numLayers="3" context="MainContext"/>
    <Torso layer="3" numLayers="3" context="MainContext"/>
    <Face layer="6" numLayers="0" context="MainContext" Tags="scope_face"/>
    <Weapon layer="0" numLayers="2" context="WeaponContext"/>
  </ScopeDefs>
</ControllerDef>
```

The controller definitions file can include a number of different tags:

- **Tags** – References the scope's tag definition (*Tags.xml) file.
- **Fragments** – References the scope's FragmentID definition (*Actions.xml) file.
- **FragmentDefs** – Contains one entry for each FragmentID specified in the FragmentID definition file. For each FragmentID, a scopes attribute defines the scopemask, optional flags attributes that
control fragment play, and the override attribute that overrides the scopemask when certain tags and frag tags are matched.

- **Subcontexts** – Lists all subcontexts available.
- **ScopeDefs** – Defines the scopes and scope contexts used. Each element defines a scope.

## Creating and Editing Scope Contexts

A scope context defines which entity, character, and animation database to use. You can use the same scope context for multiple scopes. Because every scope is attached to a scope context, at least one scope context is needed for each character.

Scope context properties may change during runtime, so it is possible to swap the entity, character instance, or animation database at any time. You can use this technique to change weapons or attach other characters to the player during a synchronized animation, for example.

Scope contexts are defined in the controller definition file (*ControllerDefs.xml*).

The implementation of the animated character game object extension is hardcoded to support the scope contexts Char1P, Char3P, and Audio.

The controller definitions file must use the Char3P scope context when using Mannequin object or the actions and layers will not play, as shown below:

```
<ScopeContextDefs>
  <Char3P />
</ScopeContextDefs>
<ScopeDefs>
  <FullBody layer="0" numLayers="3" context="Char3P"/>
```
Using Scope Masks

A scope mask is the set of scopes that a fragmentID runs on. Each fragmentID has a scope mask associated with it, as defined in the Controller Definition File using the FragmentID Editor. When an action requests a fragmentID, the action owns the scopes in the FragmentID scope mask and starts playing fragments on these scopes.

For example, a Fire Weapon fragmentID could have a scope mask containing the weapon scope for animating the weapon as well as the torso scope. It doesn't need to contain the other scopes of the character because it can control the torso independently of the rest of the body using additive and partial-body animations.

Playing Fragments on Scopes (Actions)

Scopes are defined portions of a character's body where fragments are triggered and played. By playing different sequences of animations (fragments) on specific parts of a character's body (scopes), realistic movements and motions can be achieved. This process is called a mannequin action.

One fragment can play on the full-body scope (walking), while another fragment plays on the torso scope (rotating), and yet another fragment plays on the head scope (looking at target), all simultaneously.

Fragments use Flow Graph nodes or game code to play on scopes.

Mannequin Tags (Animation Contexts)

When multiple fragments are assigned to a single FragmentID, such fragments are simply variations of ideas expressed in that FragmentID. With Tags, you can label fragments for more specific character contexts like crouched, shooting, or scared.

The game looks for tags based upon the state of the game character. For example, when a character is crouching, the game starts looking for fragments tagged as crouched. And when the character is using a machine gun, the game looks for fragments tagged as machineGun. If the game is looking for both of these tags at the same time, it first looks for a fragment with both tags. Next, the game looks for fragments labeled either machineGun or crouched. Finally, it looks for a fragment with an empty set of tags that acts as a fallback. Fragments with other tags such as swimming are not selected.

Multiple fragments can have the same set of tags and FragmentID. In this case, the game automatically assigns each fragment an option index. By default a random option index is chosen, but you can have the game select a specific one if needed, such a particular fragment for animation streaming. For example, if you have 20 variations (options) but want to stream in only one of them, you can override the random selection process and make sure that the specific variation you streamed in is selected.

When working with tags, it's useful to know the following terms:

- **Tag Definition** – A collection of tags.
- **Tag Group** – A mutually-exclusive set of tags.
- **Tag State** – A combination of tags, such as crouching+pistol.

Topics

- Using Tag Definitions (p. 106)
- Using Tag State Keys (p. 106)
- Using FragmentID Tags (Frag Tags) (p. 106)
Using Tag Definitions

Tag definitions define a collection of fragment tags. You use Tag Definition Editor within Mannequin Editor to create tag definitions and store them in a tag definition (*.Tags.xml) file, or you can create the tag definitions file manually in a text editor.

Each tag must have a unique name within a tag definition file. Tag definition files can include (nest) other tag definition files. To edit a nested tag definition, you manually edit the tag Definition (*.Tags.xml) file. For all other tag definitions, you can use the Tag Definition Editor, which you access from the Fragments pane.

Note that Lumberyard ignores the casing of tags.

Using Tag State Keys

A tag state is a combination of tags from a tag definition. Tag states are represented by a list of tags separated by + characters. For example crouching+pistol defines a tag state combining the tags crouching and pistol.

A game can set global tags describing the current state of the character, or the global tag state. This typically contains global state information like character type, stance, and weapon attachment for example.

The global tag state is the tags member of the ActionController SAnimationContext, which is found with IActionController::GetContext().

Study the following numbered fragment timeline screen shots to understand the use of tag state keys:

- Select the {kneeling+tired} tag state key.
- Disable the tired tag in the key.
- Note the tag state key changes to {kneeling}

The FragmentID (below the tag state key) selected is the default Idle{<default> - 0}. This fragment represents the best match for the game’s request.

For the {kneeling+tired} tag state key, select the tired tag check box again.

Now drag the {kneeling+tired} tag state key to the right in the timeline.

This simulates a situation where the game requests {kneeling+tired} after requesting the Idle{<default> - 0} fragmentID. This means that at the moment Idle is requested, the tags are not set, and the default FragmentID is selected.

The order in which game requests arrive in the Mannequin system has an influence on which fragments get selected eventually. For example, if you want to move a certain fragment around, you need to select both the FragmentID and the tag state key above it.

Using FragmentID Tags (Frag Tags)

FragmentID-specific tags, also known as frag tags, are tags that are assigned only to fragments with a specific fragmentID.

Many fragment tags don't have to be available to all fragments. For example, there might be a hit fragmentID that groups fragments containing hit reaction animations. The actual type of hit, such as
headshot or explosion would then be encoded in tags. But such tags are only useful in the context of the hit fragmentID, so such tags are considered fragmentID-specific.

Frag tags are created by creating a new tag definition using the Tag Definition Editor in Mannequin Editor. This new tag definition is then assigned to a FragmentID using the FragmentID Editor.

Each fragmentID can have only one tag definition containing frag tags, but for more complicated cases you can import other tag definition files hierarchically from the main tag definition.

Frag tags are stored in separate tag definition files that are linked to from the fragmentID definition file as sub-tag definitions.

Assigning Fragment Tags

Tags are added to fragments to limit which fragments can get selected. For example, a "tired" tag can be assigned to a fragment so it only gets selected when the character is tired. Or for example, other fragments can be assigned "kneeling" or "standing" tags to create different "stance" variations for the same animation.

For this example, "stance" is considered a tag group. Some tags are inside tag groups, some other tags, like "tired", are not. Putting tags in a group ensures sure you can only select one of the tags in the group at the same time. So a character cannot be both "standing" and "kneeling" at the same time for example, but can be both "kneeling" and "tired". The various tags within a tag group are called tag options.

The order in which the fragments are listed in the Fragments browser reflects the order in which they are selected. If there are multiple equivalent matches, the first match in the list is selected. For example, you might have a tag called "tired" and a tag called "scared." You have one fragment tagged "tired" and another fragment tagged "scared." The game looks for a fragment for a character that is both "tired" and "scared." If "tired" and "scared" have the same priority, it is undefined which fragment is chosen, but the Mannequin Editor shows you the fragments in the selection order.

Mannequin Animation Transitions

Animation transitions blend together multiple fragments in a specified sequence. Specifically, game code requests multiple FragmentIDs sequentially, and those associated fragments need to be blended together. With Mannequin, you can specify complex transitions between the fragments, such as specifying exactly how individual layers within fragments are combined, or the ability to add new procedural clips in between existing animation clips.

Transitions are stored with their associated fragments in the XML-based animation database .adb file. The FragmentBlendList element contains the transitions, as the following shows.

```
<FragmentBlendList>
  <Blend from="" to="idlePose">
    <Variant from="" to="">
      <Fragment selectTime="0" enterTime="0">
        <AnimLayer>
          <Blend ExitTime="0" StartTime="0" Duration="0"/>
        </AnimLayer>
      </Fragment>
    </Variant>
  </Blend>
</FragmentBlendList>
```

Topics

- Creating and Editing Transitions (p. 108)
Creating and Editing Transitions

Without transitions, a character’s motion snaps between two fragment clips using the default blend time specified for the beginning of the second fragment. Add custom transitions for more realistic motion.

To add a new transition between two fragments
1. In Mannequin Editor, on the Transitions tab, click New.
2. Select the first fragment in Fragment ID From and select any associated tags.
3. Select the second fragment in Fragment ID To and select any associated tags.
4. View the new transition in the Transitions list and the Transitions Preview timeline window. The transition is colored orange.

By default, the transition duration is the default blend time. You can easily change the transition duration time.

To change transition duration time
• In the Transitions Preview timeline window, drag the vertical divider line to the right or the left for the transition to increase or decrease the duration.

To add an animation to a transition
1. In the Transitions Preview timeline window, double-click after the start of the orange transition block.
2. Select an animation clip in Animation.
3. Drag the new clip in the Transitions Preview timeline window until the blend time of the second clip overlaps with the end of the new transition clip.
4. Right-click the first fragment and click Insert Transition.

The default transition behavior for a non-looping fragment is to wait until the end of the fragment to begin. You can adjust a transition so that a second fragment does not start playing until the first fragment is finished playing (and not immediately when requested).

To delay transition start time
1. Select any key on the transition.
2. Under Transition Properties, adjust the value of the Earliest Start Time property. This value is relative to the end of the previous fragment.

Setting Transition Parameters

There are two broad types of parameters that can be edited using Mannequin Editor – action parameters and motion parameters.

Action Parameters

These are parameters the game uses when playing actions and procedural clips. Some examples include providing a target position when aligning an entity, providing a weight value when fading an animation in or out, or providing a sound parameter.
All action parameters have a name and a value.

**Motion Parameters**

These are parameters that get passed to the blend spaces (bspaces) parametric animation system. You can preview how these parameters influence animation by adding keys for them on the Params track in Mannequin Editor.

**Cyclic Transitions**

To set up a transition from a looping or parametric animation, set the transition Select Time value relative to one cycle (or segment) of the animation clip. If the fragment changes duration, the time would automatically adjust in the proper proportion. You do this by selecting Cyclic Transition under Transition Properties. This turns the select time into a value between 0 and 1 instead of a value in seconds.

The following fragment shows:

- The first fragment is looping.
- Cyclic Transition is selected
- The select time is 0.5, and this translates into 50% along the cycle. Also displayed is the range of the select time, in this case it runs all the way to the end of the cycle. After that the second transition with select time of 0 is selected.

Unless marked as being locked, cyclic transitions always trump the previous fragment, regardless of action priority. The Earliest Start Time value is thus effectively ignored.

It is possible to delay transitions in an animation using the Earliest Start Time value. By default, this value is relative to the end of the previous fragment. For fragments with no clear ending, such as fragments with looping end clips, this is handled by "locking" the cycling so that transitions are triggered when preceding animations are a certain portion of their run cycle.

In this case, select both Cyclic Transition and Cyclic Locked. This enables the Earliest Start Time value to be stored cyclically in that the time restarts at zero after each cycle.

**Mannequin Animation Actions**

An action is a programmatic construct that used to control animations and synchronize them with the game, combining game code with simple high-level animation control.

When an action is installed, it "owns" one or more scopes and can request FragmentIDs to play on those scopes. Each scope can be controlled by only a single action. Many actions can be running in parallel as long as they all control different scopes.

Although each action can only request one FragmentID at a time, it can nonetheless sequence multiple such requests in a row. If you want to implement an animation state machine, either you queue multiple actions that each push a single FragmentID and you handle the state machine externally, or you queue a single action that has an internal state machine that requests the appropriate FragmentIDs. The latter is typically how Lumberyard handles basic locomotion state machines.

The Mannequin ActionController (IActionController) is the root object that controls a character mannequin. You configure it in a controller definition (*ControllerDefs.xml) file, which defines the FragmentIDs, scopes, and scope contexts. It also installs actions onto scopes and holds the global tag state.

**Topics**

- Creating Mannequin Actions (p. 110)
Creating Mannequin Actions

You may want to create a new action class or simply use a generic one for simple cases.

With this constructor, you can do the following:

- Set the relevant FragmentID, which is the first FragmentID that gets requested.
- Set any FragmentID-specific tags (frag tags).
- Set the action priority, which is used to manage overlapping actions and actions that want to own the same scope. Higher numbers indicate higher priority.

The following shows a sample code snippet that creates an action that plays the Idle FragmentID.

```cpp
const FragmentID idleFragmentId = m_pAnimationContext->controllerDef.m_fragmentIDs.Find( "Idle" );
const int actionPriority = 0;
IActionPtr pAction = new TAction< SAnimationContext >( actionPriority, idleFragmentId );
```

Mannequin Action Queuing

Actions are queued onto the target Mannequin ActionController(IActionController), which is the root object that controls the character mannequin.

For actors, the ActionController is accessible via the AnimatedCharacter extension (IAnimatedCharacter::GetActionController()).

A queueing statement looks like the following: `pActionController->Queue( pAction );`

This is a priority queue where higher priority actions are selected first. For each frame, the Mannequin system checks whether queued actions can be installed on the applicable scopes. Specifically, the FragmentID is retrieved and associated scope mask is determined.

If an action has higher priority than all the actions currently owning those scopes, it is installed immediately and skips any waiting times in transitions. This is called trumping. Otherwise the candidate action waits for those actions to finish or for a suitable transition to gracefully stop the current action.

When an action gets selected from the queue, it gets installed on its scopes, and its fragmentID is pushed on and updated before the next batch of animations are sent off for processing.

Actions that get pushed away are stopped unless the interruptible flag is set, in which case they get pushed back to the queue and return when they can. The interruptible flag is typically used for actions controlling Movement or Idling actions. These are low-priority interruptible actions that run by default on certain scopes but get pushed back by more specific actions.

Using Action Subcontexts

Subcontexts are a way for programmers to explicitly refer to a single logical role (out of multiple such roles) when requesting an action. Subcontexts are a convenience when dealing with FragmentIDs whose scope mask encompasses multiple scope contexts, where each context refers to a different role. For example, a car could have multiple seats, each one with its own scope and unique associated tag. Subcontexts do not affect fragments but rather provide additional contextual information when dealing with actions that involve multiple independent scope contexts.
Subcontexts are defined in the controller definition (*ControllerDefs.xml) file. Each subcontext has a unique name and exposes a scope mask and global tag state. Using the car example, the following code shows how the car’s controller definition file could define different subcontexts for different seats, each seat having its own set of scopes.

```xml
<SubContexts>
  <Driver scopeMasks="Driver+DoorDriver" tags="Driver"/>
  <Passenger scopeMasks="Passenger+DoorPassenger" tags="Passenger"/>
</SubContexts>
```

Upon entering the car, a character typically gets enslaved to either the Driver or Passenger scope context. When requesting a FragmentID that is local to one of the seats (for entering or leaving the vehicle), the game needs to state the correct subcontexts. This is done by requesting the subcontext in a mannequin action. The following snippet shows an action installed on a subcontext:

```c++
// Driver just entered the vehicle, already enslaved to it
IActionController* pVehicleActionController;
IAction* pEnterVehicleAction;

// ...

// Queue the "EnterVehicle" FragmentID with the suitable SubContext
pEnterVehicleAction->SetFragment(EnterVehicle);
pEnterVehicleAction->SetTagContext(isDriver ? Driver : Passenger); // Change SubContext based on which role the enslaved character is supposed to have
pVehicleActionController->Queue(pEnterVehicleDriverAction);
```

This results in automatically adding the matching scope mask and global tags to the default state during the fragment selection process for this action. In this example, with the proper setup, Mannequin would then know which character and which door to animate when processing this action. As such, the FragmentID can be queried and resolved to different scope masks and ultimately fragments based on the given subcontext.

### Adding Mannequin Audio

Sound is added in the Mannequin system by inserting audio procedural clips to fragments. Sound effects can be very granular, with different sounds used for different weapons in different states of firing for example. The general process is as follows:

- Reserve a scope just for audio, and place an ATL-Trigger on it.
- Edit the scope mask to include the audio scope.
- Add a ProcLayer track for the audio scope.
- Add a procedural clip, and set the type to Audio.
- Set the appropriate start and stop triggers as well as other parameters to affect the sound's properties.

### Using the New Mannequin Setup Window

You can use the Mannequin system to create complex, interactive character animations and define motions that are variations on a theme. For example, you can create a family of motions based on a running theme: running while injured, running while exhausted, running slowly, and so on.

Mannequin supports animation layering, blending, additive animations, and partial body animations. Each character animation requires the following Mannequin setup:

- **Controller definition file** (*ControllerDefs.xml*) – Defines the mannequin setup.
• **Animation database file** (.adb) – Stores fragments and transitions. A fragment is a variation in a family of motions. A transition is used to move smoothly between families of motions.

• **Tag definition file** (*Tags.xml) – Stores tag definitions. You use tags to categorize fragments that you can then select during playback.

• **Fragment ID definition file** (*FragmentIDs.xml) – Stores fragment ID definitions. Fragments that share a fragment ID are grouped into families.

Lumberyard provides a setup tool that generates these files for a mannequin setup. For more information about these file types and how they are used, see Mannequin System Files (p. 82).

**Creating Mannequin Setup Files**

You can use the **New Mannequin Setup** window to create your mannequin setup and generate the necessary files.

**To create mannequin setup files**

1. In Lumberyard Editor, choose **Tools, Animation, Mannequin Editor**.
2. In the **Mannequin Preview** dialog box, click **Create**.
3. In the **New Mannequin Setup** dialog box, type a name and then click **OK**.

   The setup name should reflect your character or class of characters. The setup tool creates .xml files with the name you specify as the default file name.

4. In the **New Mannequin Setup** window, do the following:
   a. Review the provided information for the following:
      • **Controller Definitions File**
      • **Tags File**
      • **Actions File**
      • **Preview ADB**
      • **Preview File**

      These settings represent the .xml files that the setup tool will create. You can modify the file names; however, we recommend using the default file names.
   b. For **Preview Model**, click the browse (..) button to choose your character definition file (.cdf). You must choose a .cdf file in order to generate the preview files.
5. Click **Create**.

6. In the **Output Files** dialog box, click **Yes** to confirm the file names and directory paths. The setup tool creates your mannequin files and loads the preview file in the editor.

**New Mannequin Setup Properties**

In a mannequin setup, you can modify the properties and attributes for controller definitions and for the main preview character.
Controller Definitions

A controller definition includes all of the data used to define a mannequin setup.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context Definitions</td>
<td>Specifies the category of the motion set. For example, Main, Player, Alien, Enemy, etc. You can add context definitions by clicking the + button. Click the X button to delete a context definition.</td>
</tr>
<tr>
<td>Context Name</td>
<td>Specifies the name of the context definition.</td>
</tr>
<tr>
<td>Scope Definitions</td>
<td>Categorizes the parts of a character. For example, you can create categories for the full body or isolate an animation set based on the torso, arms, legs, etc.</td>
</tr>
<tr>
<td>Scope Name</td>
<td>Specifies the name of a scope definition entry.</td>
</tr>
<tr>
<td>Start Layer</td>
<td>Specifies which of the 16 animation layers that Lumberyard supports per character to include in your scope definition. This property represents the first of the range of layers that the scope uses.</td>
</tr>
<tr>
<td>Layer Count</td>
<td>Specifies the number of layers to assign to the scope. This property represents the maximum number of animation layers for your fragments in the Mannequin Fragment Editor timeline.</td>
</tr>
<tr>
<td>Controller Definitions File</td>
<td>Defines a mannequin setup. This file is used by the Mannequin Editor and your game and is typically referred to from the character Lua file and Mannequin preview file. By default, the Mannequin setup tool generates a new .xml file.</td>
</tr>
<tr>
<td>Tags File</td>
<td>Stores tagging information in an .xml file. You can use tags in Mannequin to group families of animations. You can edit this file at any time using the Mannequin Tag Definition Editor. By default, the Mannequin setup tool generates a new .xml file. You can reference an existing file by clicking the browse (..) button and navigating to an existing tags file.</td>
</tr>
<tr>
<td>Actions File</td>
<td>Stores fragment ID definitions. This file is also called the FragmentID Definition File and is used by the Mannequin Editor and your game. The controller definition and animation database files refer to this file. By default, the Mannequin setup tool generates a new .xml file. You can reference an existing file by clicking the browse (..) button and navigating to an existing actions tags file.</td>
</tr>
</tbody>
</table>

Main Preview Character

The mannequin preview uses the controller definition data to test your mannequin setup while authoring.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character Name</td>
<td>Specifies the name of the character in the preview window.</td>
</tr>
</tbody>
</table>
Synchronizing Multiple Characters

Synchronizing multiple animated characters is a common task. Practical examples include animating a weapon in sync with a character's body when reloading or firing, or synchronized actions across multiple characters, such as for stealth kills.

This can be achieved with Mannequin through the use of scope contexts and the concept of coupling or enslavement.

The first step required to synchronize a secondary character with a primary one is to add an extra scope and scope context in the host character's Controller Definition *ControllerDefs.xml* file. The secondary character is then attached to the newly-created scope context. The following is an example ControllerDefs.xml file:

```xml
<ControllerDef>
  ...
  <ScopeDefs>
    <FullBody1P layer="0" numLayers="3" context="Char1P"/>
    ...
    <FullBody3P layer="0" numLayers="3" context="Char3P"/>
    ...
    <Weapon layer="0" numLayers="3" context="Weapon"/>
    ...
    <AttachmentTop layer="0" numLayers="3" context="attachment_top"/>
    <AttachmentBottom layer="0" numLayers="3" context="attachment_bottom"/>
    <SlaveChar layer="0" numLayers="3" context="SlaveChar tags="slave"/>
    <SlaveObject layer="0" numLayers="3" context="SlaveObject tags="slave"/>
  </ScopeDefs>
</ControllerDef>
```

This example shows seven scopes using seven different contexts, which means that fragments can be synchronized for up to seven different characters.

**Parameters**

<table>
<thead>
<tr>
<th>Scope</th>
<th>Scope Context</th>
<th>Layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>FullBody1P</td>
<td>Char1P</td>
<td>0, 1, 2</td>
</tr>
</tbody>
</table>
Using Flow Graph with Mannequin

Some Mannequin system functionality is available using the `Actor:PlayMannequinFragment` and `Actor:PlayMannequinFragment` Flow Graph nodes.

The `Actor:PlayMannequinFragment` node looks for a fragment to play using the provided FragmentID and TagState. This fragment is in a Mannequin Action and queued with the given priority. The `Actor:PlayMannequinFragment` node can also stop this action using the `ForceFinishLastQueued` input, or pause/resume the entire Mannequin ActionController.

Some guidelines and best practices for using this node include the following:

- Make sure that querying fragments do not conflict with AI, player, or game logic if the entity being targeted is also driven by other game code.
- Select priority based on what you want to interrupt. Movement fragments run at priority 4, hit reactions at priority 5, and death reactions at priority 6.
- You cannot start an action on one node and stop it with another node. Actions are not shared across nodes.

Debugging Mannequin System Issues

Lumberyard offers a number of methods for debugging Mannequin system issues. In addition to the ones listed below, you can also analyze an error report.

Topics

- Using Console Variables (p. 116)

Using Console Variables

Use the following console variables for debugging the Mannequin system.

- `mn_allowEditableDatabasesInPureGame` mn_DebugAI – Do not store editable databases.
**Physicalizing Characters (Ragdoll)**

Characters can have two skeletons: the main (alive) skeleton and the ragdoll (injured or dead) skeleton. Use Lumberyard to physicalize the ragdoll skeletons that you created in your DCC tools.

The ragdoll skeleton is what the main skeleton swaps to when it is inflicted with enough damage. It can be more simplified than the main skeleton and often uses capsules on limb joints for more accurate simulation. The two skeleton types have similar physics mesh for joint limits and spring attributes. Any differences are used to fine-tune the ragdoll simulation.

**Topics**

- Ragdoll Best Practices (p. 117)
- Ragdoll Skeleton DCC Setup (p. 117)
- Creating Joint Mesh Proxies (p. 118)
- Using physParentFrames (p. 121)
- Applying Simulation Settings to Ragdoll Joints (p. 122)
- Lumberyard Proxy Tool (Experimental) (p. 125)
- Adding Mesh Proxy Materials (p. 126)
- Ragdoll Physics (p. 126)

**Ragdoll Best Practices**

There are a few guidelines and best practices to follow when creating physicalized ragdoll characters:

- To help with performance, use simplified geometry (for example, boxes or capsules) for your phys mesh proxies whenever possible.
- In Maya, the physics skeleton does not use the SceneRoot node to determine the up axis in the scene.
- The root joint and first or highest hierarchical ragdoll joint, such as the pelvis or hip, must be oriented to match z-up.
- For self-collision to work correctly, use the following naming conventions (case-sensitive) for your skeleton joints:
  - Pelvis
  - Spine
  - Head
  - UpperArm
  - Forearm
  - Thigh
  - Calf

**Ragdoll Skeleton DCC Setup**

You can set up your ragdoll skeleton in a DCC tool such as Maya or 3ds Max.
Creating Joint Mesh Proxies

To setup a ragdoll skeleton

1. Create phys mesh proxies for the main joints that need to ragdoll. They should match the orientation of their respective joint. The first or highest hierarchical ragdoll joint and proxy (such as the pelvis and hip) should have the z-up orientation.

2. Name the physics mesh proxies based on the joint they represent and with the appropriate suffix. Examples:
   - For Maya, a joint named def_l_thigh would have a physParentFrame named def_l_thigh_phys.
   - For 3ds Max, a joint named Bone C SpineA would have a physParentFrame named Bone C SpineA Phys.

3. Create any necessary PhysParentFrame groups or nodes for joints that need to rotate more than the y-axis limit range of -90 to 90 degrees and name them based on the joint they represent with the appropriate suffix. For example:
   - For Maya, a joint named def_l_thigh would have a physParentFrame named def_l_thigh_physParentFrame.
   - For 3ds Max, a joint named Bone C SpineA would have a physParentFrame named Bone C SpineA Phys ParentFrame.

4. Assign each physics mesh proxy and physParentFrame to a parent. If using a physParentFrame, nest the physics mesh proxy under the physParentFrame, and the physParentFrame under the joint it belongs to. If there is only the physics mesh proxy, nest the proxy under the joint it belongs to.

5. Create and assign physics mesh materials to the appropriate physics mesh as needed. For example, a material for the left arm assigned to physics mesh proxies that are part of the left arm.

6. Add rotation limit values to skeleton joints that will be used in the ragdoll. In addition, you must add some rotation limit information to joints that do not have a physics mesh proxy but that are in the hierarchy of the ragdoll joints. For example, a clavicle needs some rotation limits even though it typically does not have a physics proxy mesh.

7. Export the skeleton's .chr file and the material group or multimaterial that contains the physics mesh materials.

Creating Joint Mesh Proxies

To help define body masses for character physics and collisions, you need to use a joint mesh proxy. To create a joint proxy mesh, observe the following guidelines:

- Model meshes around the geometry that needs to be detectable. Meshes with lower polygon counts perform better.
- Create meshes that use a generalized area instead of getting too granular with a physics mesh per joint to help with performance. For example, you could simplify a biped chest physics mesh proxy to cover the area of multiple spine joints instead of a proxy for each spine joint.
- Nest a proxy mesh under its corresponding joints. The proxies can then be exported with the .chr (skeleton) files.
- Proxy mesh naming must match the name of the joint it gets nested under with the addition of the following suffix:
  - For Maya, add the _Phys suffix (not case sensitive)
  - For 3ds Max, add the Phys suffix (not case sensitive)
- To designate meshes as proxies, assign a material to them and changing the material type to Proxy No Draw. As a best practice, keep your ragdoll physics mesh materials in a separate material group.
In the following figure, two different proxy materials are applied to the proxy mesh. Lumberyard uses the different materials to detect different parts of the body. In this case, the separate head material allows the engine to distinguish if the head is interacting with an object, as opposed to the rest of the body interacting with an object. For example, if a character gets hit in the head, you might want a special animation reaction to play, as opposed to the character getting hit in the body.
Ragdoll characters can have more proxy materials to define specific areas of the body. For example, you can have a proxy material for the head, spine, hip, left arm, right arm, left leg, right leg, left foot, and right foot.
Using physParentFrames

If a character joint needs to rotate beyond the -90 to 90 degree y-axis limit range, a physParentFrame node can be created. For example, if you need a joint that needs to rotate in the -120 to 120 degree y-axis limit range, you would create a physParentFrame node or group for the joint. The physParentFrame node could have a -50 to 50 degree y-axis limit range and the phys mesh could have a -70 to 70 degree y-axis limit range to have the joint combine to a -120 to 120 degree y-axis limit range.
You do not need to use another mesh for the `physParentFrame`. You can use a group node (for Maya) or dummy (for 3ds Max) as the `physParentFrame`.

In Maya, the naming convention for a `physParentFrame` is `joint name + _physParentFrame` suffix (not case sensitive).

In 3ds Max, the naming convention for a `physParentFrame` is `joint name + physParentFrame` suffix (not case sensitive).

You nest the `physParentFrame` under the joint and nest the physics mesh under the `physParentFrame`.

### Applying Simulation Settings to Ragdoll Joints

You will want to apply rotation limit values to your joints used for ragdoll depending on what ranges you want the joint to have. The default values are set to 0 degrees, but the range is -90 to 90 degrees, with the lowest range value being set in the rotation minimum and the highest range value being set in the rotation maximum. If you need more than the -90 to 90 degree y-axis range, you will need to create a `physParentFrame` node. For more information, see Using `physParentFrames` (p. 121).

Any joint that has 0 degrees set for all the rotation limit ranges and is not in the Active and Limited states will be treated as non-physicalized. For this reason, it is a good idea to set some random limit ranges on joints that do not have a phys mesh proxy, but have child joints that do have a phys mesh proxy so the rest of the chain will still be physicalized. You will not need to enable the Active and Limited states for these joints either. For example, if you have a clavicle joint with no phys mesh proxy that is the parent of your shoulder joint that had a phys mesh proxy, you will want to add some values to the clavicle joint so the shoulder will still exhibit ragdoll behavior, but do not set any of the rotations in the Active/Limited states.

**To apply simulation settings using Maya**

1. Open Maya and select the root joint for your skeleton.
2. Select **Lumberyard Tools** from the Lumberyard Shelf.
3. Select **Tools, Add Attributes**.
4. In the **Attribute Editor**, scroll down to the **Extra Attributes** panel for the root joint.

   The panel shows the ragdoll simulation settings that have been applied. This is true for every joint in the hierarchy.

   ![Extra Attributes panel](image)

5. Apply the desired simulation values for your ragdoll skeleton joints.

   Select the **Rot Limited** check boxes for your coordinates to limit rotation to the specified values. Clear the check boxes for unlimited rotation.

6. Place the lowest range value in the **Rot Limit Min** field and place the highest range value in the **Rot Limit Max** field for the x-, y-, and z-axes. For example, a joint in the -70 to 0 degree range for the y-axis would have -70 in the **Rot Limit Min** (second-column) field and 0 in the **Rot Limit Max** (second-column) field.

   **Note**
   For Maya users, there is one exception to the **Rot Limited** check boxes in the case of the pelvis/hip joint. You will want to apply some values for the **Rot Limit Min** and **Rot Limit Max** fields for the pelvis/hip joint, but keep the **Rot Limited** check boxes unchecked.

7. After simulation settings have all been applied, export the character skeleton `.chr` file and the material group or multi-material that contains the phys mesh materials.

8. In Lumberyard, use the character skeleton `.chr` file as part of a `.cdf` as normal for character assembly.

9. Open Geppetto and preview your phys mesh proxies by enabling **Display Options, Physics, Physical Proxies**, and view **Ragdoll Joint Limits**.

To test your ragdoll, use either the Ragdoll component entity or the legacy DeadBody entity. For more information, see **Ragdoll Physics (p. 126)**

**To apply simulation settings using 3ds Max**

1. Open 3ds Max and select any skeleton joint.
2. Click the **Hierarchy** tab.
3. Click the **IK** button under the name of your joint.
4. Scroll down to the **Rotational Joints** panel.
5. Apply the desired simulation values for your ragdoll skeleton joints.

When using the X, Y, Z rotations for ragdoll, enable the **Active** and **Limited** check boxes for the axes you are using and disable the **Active** check box for the axes you are not using.

Place the lowest range value in the **From** field and place the highest range value in the **To** field.

6. Set **Damping** to 1 for an active axis.

7. After simulation settings have all been applied, export the character skeleton `.chr` file and the material group or multi-material that contains the phys mesh materials.

8. In Lumberyard, use the character skeleton `.chr` file as part of a `.cdf` as normal for character assembly.

9. Open Geppetto and preview your phys mesh proxies by enabling **Display Options, Physics, Physical Proxies**, and view **Ragdoll Joint Limits**.

To test your ragdoll, use either the Ragdoll component entity or the legacy DeadBody entity. For more information, see **Ragdoll Physics (p. 126)**

**Other Parameters**

The **Spring Tension** parameter controls the stiffness of an angled spring at a joint. The default value of 1 means the acceleration of 1 radian/second^2 (1 radian = 57 degrees).

The **Damping** parameter controls how loose the joint will be in the ragdoll simulation. The default value of 1 is recommended because it corresponds to fully-damped oscillations for the joint.
Creating individual meshes for each character body part can be time-consuming. The Lumberyard Proxy Tool automates the process of building simple joint proxy meshes and adding materials.

**Note**
This tool is in the experimental phase of development.

**To create a joint proxy**

1. In the Maya scene, select a joint you want to add a proxy to and then choose **Add Joints**.
2. Use the following controls to adjust parameters as needed:
   - **Width** – Width and depth of the proxy
   - **Shape** – Shape of the proxy; options are **box**, **capsule**, and **sphere**
   - **Orient** – Orientation axis of the joint as it points to its child
   - **Material name** – Name of the proxy material
3. Choose one of the following:
   - **Create Proxies (Additive)** – Creates the joint proxies
   - **Create Proxies (Replace)** – Deletes all current proxies before creating the new joint proxies
Adding Mesh Proxy Materials

For the materials to be used for the ragdoll physics mesh proxies, you assign a surface type to each submaterial. The shader type should already be set to **Nodraw** if the materials were exported from your DCC with the **Proxy No Draw** material physics type. But note that the **Surface Type** field in the Lumberyard **Material Editor** will be empty.

You can choose to set **Surface Type** to a few options depending on whether you need to detect specific physics mesh proxies or if you do not need any additional special behavior. The **nodraw** type is a good default if you only want to use physics mesh proxies for your ragdoll skeleton. Otherwise, you can use the following settings:

- arm_left
- arm_right
- foot_left
- foot_right
- hand_left
- hand_right
- head
- leg_left
- leg_right
- torso

**Ragdoll Physics**

You can add physics to your ragdoll skeleton using either the **Ragdoll** component entity or by using the legacy **DeadBody** entity.
Using the Ragdoll Component Entity

Using the Ragdoll component entity is an easy way to test out your ragdoll asset. This requires that you have created a .cdf file using the character's skeleton .chr file that was exported from your DCC with ragdoll simulation attributes on the joints and with physics mesh proxies.

To set up a Ragdoll component entity

1. In Lumberyard Editor, right-click in your level and select Create new component entity.
2. Click Tools, Entity Inspector to view your component entity's settings, if the window is not open already.
3. In the Entity Inspector window, click Add Component.
4. Choose Physics, Ragdoll.
   
   A Skinned Mesh component is automatically added to the Ragdoll component on your entity because it is required for the ragdoll component.
5. Under the Skinned Mesh component, expand Rendering.
6. For the Skinned asset parameter, click the (...) icon and locate your character's .cdf file for the ragdoll and assign it to the entity.
   
   You can also use the bidped.cdf file located for the SamplesProject at Objects/Tutorials/Biped/ to test if you do not have a ragdoll character.
7. Under the Ragdoll, select the Enabled initially check box.
8. Click AI/Physics at the bottom of Lumberyard Editor to view the ragdoll physics for your character. Click the button again to reset the character.
9. Make additional adjustments to your Ragdoll component settings. For more information, see Ragdoll.

Note

To change the character joint rotations, you must change the simulation values on the skeleton in your DCC tool and then reexport the skeleton.

Using the (Legacy) DeadBody Entity

You may wish to use the legacy DeadBody entity for your ragdoll skeleton instead of using the Ragdoll component. The DeadBody entity is located on the Rollup Bar under Entity, Physics.

When you use the DeadBody entity, ragdoll skeletons may collapse in unpredictable ways. To counter this, use the following settings in the PhysParams and Properties panels in Rollup Bar:

- ExtraStiff = 1 (enabled)
- Mass = 80
- Stiffness = 100

When you use the DeadBody entity, the ragdoll skeleton has the following characteristics:

- The ragdoll skeleton bones act as switches, activating physicalization of the corresponding bone in the main skeleton.
- The IK limits and dampening that are used in the physics mesh are read and used in ragdoll physics to limit and dampen the movement of any given joint.
- Each node in the ragdoll skeleton stores physical properties for its corresponding bone in the deforming hierarchy, as stored in the physics bone IK properties.
• The ExtraStiff parameter turns off constraints and attempts to maintain shape by pulling the bones toward an animation pose.

**Fall-and-Play Movement**

Fall-and-play movement is activated when a character is a ragdoll (that is, has the RelinquishCharacterPhysics value) with a greater than zero stiffness. This activates angular springs in the physical ragdoll that attempt to bring the joints to the angles specified in the current animation frame. When a character is still a ragdoll you can also turn the stiffness off with a GoLimp method.

The character tries to select an animation internally based on the current fall-and-play state. If there are no or very few physical contacts, the animation shows falling. Otherwise, the animation is the first frame of a standup animation that corresponds to the current body orientation.

Whenever there is an animation with a name that starts with Standup_, it's registered as a standup animation. Standup is initiated from outside the animation system through the appropriately named function. During the standup, the character physics is switched back into an alive mode with the final physical pose blended into a corresponding standup animation. This animation is selected from a standup animation list that best matches this pose.

You can control which type to use by _CSkeletonPose::SetPAnimGroup_ methods. At run time, Lumberyard checks the most similar standup animation registered to the current lying pose and starts blending.

**Using Inverse Kinematics (IK)**

Inverse kinematics (IK) involves calculating the rotations of the joints in a character skeleton so that a specific part of the skeleton (the end effector) reaches a defined target point. Use IK when an animation requires a terminating joint to be placed precisely. All IK systems must be defined in the character's .chrparams file.

Lumberyard's animation system processes forward kinematics (FK) and IK tasks in the following order:

1. Aim IK and look IK
2. Animation-driven IK
3. Foot IK and ground alignment
4. Limb IK
5. Individual joint overrides

**Topics**

- Aim IK (Aim Poses) (p. 128)
- Look IK (Look Poses) (p. 142)
- Animation-Driven IK (p. 145)
- Foot IK and Ground Alignment (p. 146)
- Limb IK (p. 147)

**Aim IK (Aim Poses)**

When creating a game, a commonly required movement is a character aiming a weapon at a target location. This complex movement requires the weapon to point at a specific location, the hands of the
character to firmly hold the weapon, and the character to look through the scope at all times. In many cases, other nuances are added to the character while aiming.

Lumberyard provides a parametric directional blending system with which you can create a set of poses for characters aiming in different directions. At run time, these poses are layered on top of the animation that is currently playing. This enables the character to aim toward a point in space that is requested by the game code while retaining the style present in the original authored poses as much as possible. Characters exhibit a realistic range of motion. Note, however, that continuous, 360-degree aiming around a pivot point is not supported.

You can call aim IK from the Flow Graph editor, Track View editor, the AI system, or engine code.

Topics
- Setting up a Skeleton (p. 129)
- Setting Up an Animation File (p. 132)
- Setting up a .Chrparams File (p. 135)
- Testing and Debugging Aim IK (p. 139)

Setting up a Skeleton

The system requires certain joints to determine where a character is aiming. You can use existing joints in the skeleton or add extra joints to improve your setup.

Note
The joint names that are referenced for the attributes in the .chrparams file should match the names of the joints in your skeleton. The joint names don't have any specific naming requirements.

You can add the following types of joints.

- ParameterJoint – A joint that indicates the aim direction, with the y-axis forward.
• **StartJoint** – A joint that indicates the positional center of the aim. Orientation is not important because only position information is used from this joint. For more stable results, use a less animated joint that is less influenced by other joint animation. For example, use a joint that is connected to the root joint (the first skeleton joint, which is placed at origin 0,0,0).
• **ReferenceJoint** – (Optional) A joint that indicates the forward direction of the character with the y-axis pointing forward. If a value is not specified, the joint at index 1 (usually the pelvis) is used. This joint is primarily used for characters in cinematics because they might have an offset on top of the root joint. In this case, don't use the root joint as the reference joint. Instead use the joint that acts as the translation or position driving joint to determine the point that is underneath the character.
Setting Up an Animation File

For a character aiming in several directions, the system requires a certain number of poses that can be blended together to achieve poses in any intermediate direction. For best visual results, we recommend creating 15 poses, although 9 poses might be enough in many cases. If you provide 9 poses, the system extrapolates from these poses to create 15 poses.
When you export the poses, they become a single animation file, with one pose per frame. Part of the file name should match the AnimToken provided in the .chrparams aim IK definition. For example, if the AnimToken parameter is set to AimPoses, the name for the animation file that contains the aim poses could be Troop_AimPoses. Naming this file is important because Lumberyard uses the name that is assigned in the AnimToken as a filter. That way Lumberyard can properly read and compile the animation data as an aim pose and not a regular animation.

When creating aim poses, you can use an underlying animation pose, such as standing idle, as a starting point. Aim poses that are created from a starting animation are applied on top of similar animations. If the underlying animation that is currently playing for a character is different enough—for example, crouching—you might need to create aim poses for that specific case to achieve better quality animations.

The order of the poses in the animation is also important. Reference the grid visual in the following images for the top, middle, and bottom rows.
As you create poses, keep in mind the following. This information assumes that you start on frame 1 for your aim poses animation.

**Note**

Although the aim pose might appear unnatural, try to make the poses as extreme as possible. You can then set limits using the game code.

- The top row in the grid should have the character aiming upward. If you're using 15 aim poses, the top row comprises frames 1–5. If you're using 9 aim poses, the top row comprises frames 1–3.
  
  ![Top Row Frames](image)

- The first aim pose frame in the top row (frame 1) should have the character aiming up and to the right of the character.
  
  ![First Frame](image)

- The middle frame in the top row (frame 3 of 15 or frame 2 of 9) should have the character aiming up and forward.
  
  ![Middle Frame](image)

- The last aim pose frame in the top row (frame 5 of 15 or frame 3 of 9) should have the character aiming up and to the left of the character.
  
  ![Last Frame](image)

- The middle row in the grid should have the character aiming at eye level, with the weapon parallel to the ground. If you're using 15 aim poses, this row comprises frames 6–10. If you're using 9 aim poses, this row comprises frames 4–6.
  
  ![Middle Row Frames](image)

- The first aim pose frame in the middle row (frame 6 of 15 or frame 4 of 9) should have the character aiming at eye level and to the right of the character.
  
  ![First Middle Frame](image)

- The middle aim pose frame in the middle row (frame 8 of 15 or frame 5 of 9) should have the character aiming at eye level and forward.
  
  ![Middle Middle Frame](image)
• The last aim pose frame in the middle row (frame 10 of 15 or frame 6 of 9) should have the character aiming at eye level and to the left of the character.

• The bottom row in the grid should have the character aiming downward. If you're using 15 aim poses, the bottom row comprises frames 11–15. If you're using 9 aim poses, the bottom row comprises frames 7–9.

• The first aim pose frame in the bottom row (frame 11 of 15 or frame 7 of 9) should have the character aiming down and to the right of the character.

• The middle aim pose frame in the bottom row (frame 13 of 15 or frame 8 of 9) should have the character aiming down and forward.

• The last aim pose frame in the bottom row (frame 15 of 15 or frame 9 of 9) should have the character aiming down and to the left of the character.

When you export your aim pose animations, ensure that you include the exact frame count for your poses. For example, if you are using frames 1–15 for your aim poses, export frames 1–15 only. If you are using frames 1–9 for a 9-aim pose setup, export frames 1–9 only. Lumberyard specifically looks for exact frame count for aim poses, which is why the order of the poses is also important.

Setting up a .Chrparams File

Use Geppetto to set up and store the aim IK definition in a character's .chrparams file. You can set up one aim IK definition in each .chrparams file.

To enable the Aim IK definition in a character's .chrparams file

1. In Lumberyard Editor, choose Tools, Geppetto.
2. In Geppetto, in the Assets panel, expand Skeletons and navigate to the .chrparams file to which you want to add the aim IK definition.
3. Select the .chrparams file to load it in the Properties panel.
4. In the Properties panel, expand IK Definition.
5. Select the Aim IK check box.
6. Use the information in the following sections to set the AIM IK properties for Directional Blends, Rotation List, and Position List.
Directional Blends

The Directional Blends settings specify a combination of parameter, start, and reference joints to use for aim poses, as described in Setting up a Skeleton (p. 129). An animation is processed as an aim pose based on these settings when the AnimToken is found in the animation file name. For example, if a skeleton path contains the substring aim, the animation is considered an aim pose with aim direction set as a parameter joint, spine as a start joint, and Locator_Locomotion as a reference joint. You can define these joints based on your skeleton setup, and you can specify more than one Directional Blends.
To set up the Directional Blends settings, fill in the properties for AnimToken, Parameter Joint, Start Joint, and Reference Joint.

**AnimToken**

A substring that must match part of the name of an animation to be processed as an aim pose. The AnimToken includes the current configuration for parameter, start, and reference joints.

**Parameter Joint**

The name of the parameter joint. For information, see Setting up a Skeleton (p. 129).

**Start Joint**

The name of the start joint. For information, see Setting up a Skeleton (p. 129).

**Reference Joint**

The name of the reference joint. For information, see Setting up a Skeleton (p. 129).

**Rotation List**

The RotationList is populated with the list of skeleton joints from the .chr file that is associated with the .chrparams. The run-time code uses the joints that are enabled in the RotationList to calculate and blend the orientation of joints in the aim pose.

In this list, enable the joints that are used by all aim poses and/or multiple Directional Blends. Verify that the list is valid for all of them. The following parameters are available for each joint in the RotationList.
JointName

The name of the skeleton joint. If enabled, the aim pose uses this joint’s orientation data.

Add (Additive)

Enables additive blending for the joint’s orientation. When deselected, the default is to override blending.

Prim (Primary)

Specifies whether the joint is part of the hierarchical chain from the root joint up to the parameter joint.

Primary joints that are specified in the rotation list are typically the highest joints in the hierarchy going down the chain to the parameter joint. For example, a character’s arms are specifically positioned in an aim pose. The pelvis/hip joint, spine joints, and parameter joint are enabled as primary joints. But the clavicle joints through the arms are not considered primary because they are children further in the hierarchy from the parameter joint in the skeleton hierarchy.

Position List

The PositionList is populated with the list of skeleton joints from the .chr file that is associated with the .chrparams file. The run-time code uses the joints that are enabled in the PositionList to calculate and blend the position of joints in the aim pose.

In this list, include the joints that are used by all aim poses and/or multiple Directional Blends. Verify that the list is valid for all of them. The following parameters are available for each joint in the PositionList.
J\_JointName

The name of the skeleton joint. If enabled, the aim pose uses this joint's transform or positional data.

Add (Additive)

Enables additive blending for the joint's transform or position. When deselected, the default is to override blending.

Testing and Debugging Aim IK

You can verify that aim poses are working properly by viewing them in Geppetto with animation layers.

To view aim IK through animation layers

1. In Lumberyard Editor, choose Tools, Geppetto.
2. Click File, Open Character to load your character.
3. In the Assets panel, under Animations, select an animation to assign to the base animation layer. For best results, choose an animation that the aim poses were built from.
4. In the Scene Parameters panel, click the menu to the right of Animation Layers and choose Add. The new animation layer becomes your active layer.
5. In the Assets panel, under Animations, select the aim pose animation to assign to the new animation layer. This layers the aim pose animation on top of the base animation.
6. In the viewport, move the camera around and observe the character aiming towards the camera.
7. Under the aim pose animation layer, you can adjust the Direction, X Offset, Y Offset, and Smooth Time.
You can adjust the following parameters for your aim IK animation layer.

**Direction**

Select the direction to snap the aim IK:

- **Camera** – Provides navigation in the Geppetto viewport. The aim IK follows your camera's movements.

- **Forward** – Points the aim IK forward. You do not have direct control of the aim IK.

- **Target** – Exposes the **Target Position** parameter fields to set a location for the aim IK to point at.

**X Offset**

Applies an offset on the x-axis for the aim IK.

**Y Offset**

Applies an offset on the y-axis for the aim IK.

**Smooth Time**

Adjusts the time for the smoothness of the blend for the aim IK. The smaller the value, the more responsive the aim IK. The larger the value, the more delayed the aim IK.

Default value: 0.1.

Valid values: 0–1.

**Target Position**

Sets a location in xyz in the Geppetto viewport for the aim IK to point at. This parameter is only available if the **Direction** is set to **Target**.
Using the Aim IK Console Variables

You can use the following console variables for debugging aim IK.

- ca_DrawAimIKVEGrid – To display the grid for your aim poses, set this console variable to 1. The green rectangle shows your individual aim pose frame extremes. As you move the camera around in the Geppetto viewport, you see a pink cube move around the grid to indicate which blend of the aim poses is being used. If you don’t see a green rectangle or if you experience other issues, check the setup for the aim poses and the orientation of the joints in the skeleton in the .chrparams file.

- ca_DrawAimPoses – To display the aim IK's raycast and post the debug information for the coordinates on the aim IK grid, set the ca_DrawAimPoses console variable to 1.
Lumberyard Legacy Reference
Look IK (Look Poses)

- **ca_UseAimIK** – Enables or disables aim poses on a global level for debugging.
- **es_debugAnim EntityName** – Displays the current state of a character in the animation system during gameplay debugging. This console variable contains information on all the animations that are being played, including the aim poses that are played with the base animations. The combination of the aim pose with the base animation might explain why certain aim poses are broken, for example if the combination doesn't match.

The base layer also displays information about the blend weights and final influences of the aim IK and look IK, and whether or not the game is requesting this information.

**Look IK (Look Poses)**

Lumberyard supports parametric blending for automated look IK that you can use to make characters look at specific targets, even in different locomotion cycles. A character with look IK tries to look at the target as long as possible and then turns its head away. The spine, head, eyelids, and eyeballs are all animated to make the character look in the target direction. This functionality is useful in cutscene animations to make sure characters makes eye contact with the player.

Look IK can be called using flow graph, Track View editor, the AI system, or from code.
Topics

- Skeleton Setup (p. 143)
- .Chrparams File Setup (p. 143)
- Animation File Setup (p. 144)
- Debugging Look IK (p. 145)

Skeleton Setup

The system requires certain joints, listed following, to figure out where a character is looking toward. Sometimes you can use joints already present in the skeleton, but you might need to add some extra joints to make your setup work well. The look IK bone should be a child of the head bone. Make sure your eye bones are also children of the head bone.

- **ParameterJoint** – A value that indicates the direction looked in, with the y-axis forward.
- **StartJoint** – A value that indicates the positional center of the looking. Because only position information is used from this joint, its orientation is not important. For more stable results, consider using a joint that is not heavily animated, and that is not overly influenced by animation from other joints, such as a joint that is parented to the root joint.
- **ReferenceJoint** (optional) – A value that indicates the forward direction of the character, with the y-axis forward. When no value is specified, the joint at index 1 (usually the pelvis) is used. This joint is used mainly for characters in cinematics, because they might have an offset on top of the root joint.
- **AnimToken** – A substring that needs to be matched to some part of the name of an animation to be processed as a look pose with the current configuration for parameter, start, and reference joints.

**Note**

The joint names referenced for the attributes should match the names of the joints in your skeleton, but these names don’t have any specific naming requirements.

.Chrparams File Setup

Look IK parameters are stored in the .chrparams file, whose format is shown in the following example. You can have at most one `<LookIK_Definition>` tag block within an `<IK_Definition>` tag block. Within a `<LookIK_Definition>` tag block, you can have at most one of each of the following blocks: LEyeAttachment, REyeAttachment, PositionList, RotationList.

```xml
<Params>
  <IK_Definition>
    <LookIK_Definition>
      <LEyeAttachment Name="eye_left"/>
      <REyeAttachment Name="eye_right"/>
    </LookIK_Definition>
    <DirectionalBlends>
      <Joint AnimToken="LookPoses" ParameterJoint="Bip01 Look" StartJoint="Bip01 Look" ReferenceJoint="Bip01 Pelvis"/>
    </DirectionalBlends>
  </IK_Definition>
  <RotationList>
    <Rotation Additive="1" Primary="1" JointName="Bip01 Pelvis"/>
    <Rotation Additive="1" Primary="1" JointName="Bip01 Spine"/>
    <Rotation Additive="1" Primary="1" JointName="Bip01 Spine1"/>
    <Rotation Additive="1" Primary="1" JointName="Bip01 Spine2"/>
    <Rotation Additive="1" Primary="1" JointName="Bip01 Spine3"/>
    <Rotation Additive="0" Primary="1" JointName="Bip01 Neck"/>
    <Rotation Additive="0" Primary="1" JointName="Bip01 Head"/>
    <Rotation Additive="0" Primary="1" JointName="Bip01 Look"/>
  </RotationList>
</Params>
```
<PositionList>
    <Position Additive="1" JointName="Bip01 Pelvis"/>
</PositionList>

</LookIK_Definition>
</IK_Definition>

... 

<Params>

### DirectionalBlends section

The `DirectionalBlends` section specifies a combination of parameter, start, and reference joints to use for look poses. An animation is processed as a look pose with this specific configuration when the `AnimToken` is found somewhere in its name. For example, any animation processed for a skeleton that contains the substring `LookPoses` anywhere in its path is considered a look pose with `Bip01 Look` as a parameter joint, `Bip01 Look` as a start joint, and `Bip01 Pelvis` as a reference joint.

You can specify more than one `DirectionalBlends` section.

### RotationList section

The list in the `RotationList` section is used by the run-time code to identify the joints that contribute their orientation to look poses. Any joint not in this list will be ignored for the purposes of calculating and blending the look pose.

Primary joints should be specified at the start of the rotation list. All primary joints must appear in the list before any of their children that are also marked as primary.

LookPoses can only have one rotation list, so all joints used by all look poses should appear in this list, and the list should be valid for all of them.

- **JointName** – The name of the joint.
- **Additive** – The blend mode, where 1 is additive blending and 0 (zero) is override blending.
- **Primary** – A value that specifies if the joint is part of the hierarchical chain that goes from the root joint up to the parameter joint.

### PositionList section

The list in the `PositionList` section is used by the run-time code to identify the joints that contribute their position to look poses. Any joint not in this list is ignored for the purposes of calculating and blending the look pose.

Look poses can only have one position list, so all joints used by all look poses should appear in this list.

### LEyeAttachment and REEyeAttachment

These optional parameters specify the names of the left and right eyeball attachments. These parameters are used during skeleton post-processing to orient those attachments toward the target location. These parameters are relevant only if you use attachments for the eyes.

### Animation File Setup

The system requires a number of poses for a character looking in several directions so that it can blend between the poses to look in any intermediate direction. The system works with 9 or 15 poses. Although 9 poses might be enough for many cases, we recommend that you use 15 poses for better visual results. When you provide 9 poses, the system extrapolates from the provided ones to create 15 poses.

The poses are exported as an animation file, with one pose for each frame. Naming for this file is important. Some part of its name should match the `AnimToken` provided in the definition.
The order of the poses in the animation is also important.

When creating look poses, commonly you use an underlying animation pose as a starting point (such as standing idle). The look poses created from such a starting animation must be applied on top of similar animations. If the underlying animation currently playing for a character is different enough (such as crouching), you might need to create look poses for that specific case to achieve better quality.

Try to make the poses as extreme as possible, even though they might look unnatural. Limits can then be set using the game code. The middle pose (frame 4 of 9) needs to point forward. The other poses are centered around the middle pose. The angle between the middle pose and the remaining look poses should be approximately 70 degrees.

**Debugging Look IK**

The easiest way to verify that look poses are working properly is to look at them in Geppetto with animation layers.

**To view animation layers in Geppetto**

1. Load your character in Geppetto.
2. Start an animation, and assign it to the base animation layer.
3. In the *Scene Parameters* panel, choose *Animation Layers*, and then choose *Add*. A new animation layer is added that has no animation assigned to it yet. This layer will become your active layer.
4. Select the look pose animation to assign it to the new animation layer.
5. The look pose animation is now layered on top of the base animation. Move the camera around in the Geppetto viewport, and observe the character looking towards the camera.
6. Under the look pose animation layer, adjust the direction of aiming, offset, and time-smoothing as needed.

Set the `ca_DrawAimIKVEGrid` console variable to 1 to display the grid for your look poses. The green rectangle shows your individual look pose frame extremes. As you move the camera around in the Geppetto viewport, you will see a red cube move around the grid to indicate which blend of the look poses is being used. If you don't see a green rectangle or are running into other issues, recheck the setup for the look poses in the `.chrparams` file and the orientation of the joints in the skeleton.

You can use the `ca_UseLookIK` console variable to enable or disable look poses on a global level for debugging.

To see the current state of a character in the animation system during gameplay debugging, you can use the `es_debugAnim EntityName` console variable. Because this variable contains information on all animations that are being played, you can get information on which aim poses and look poses play with which base animations. The combination of the look pose with the base animation might explain why certain look poses look broken, for example if the combination doesn't match.

The base layer also displays information on the blend weights and final influences of the look IK and aim IK, and whether it is being requested by the game or not.

**Animation-Driven IK**

Lumberyard supports animation-driven IK that can retarget limbs on the fly and that is controlled by the animation. You begin by controlling and animating the blend weight of this IK in your DCC tool.

An additional `_IKTarget` bone and `_IKBlend` weight bone inside a character's skeleton defines the IK target and the blend weight. These weights ensure that a limb reaches a specific destination regardless of animations in higher layers that modify the skeleton. For example, you might create a weapon reload animation that always brings the character's hand to the pocket at the belt, regardless of upper body
animations rotating the torso and arms. You can also blend from one IK target to another, such as blending the left hand from a weapon to the magazine and back again.

Animation-driven IK can save memory and asset creation. For example, you can use the same aim pose for different guns by simply moving the IK target to the correct location on the new weapon.

You define the IK solver for a character inside the .chrparams file. Each entry in the file specifies which solver (2-bone, 3-bone, or CCD IK) to use with a chain of bones, the _IKTarget bone, and the _IKBlend weight bone.

You can animate both the _IKTarget bone and the _IKBlend weight bone. If the _IKBlend weight bone indicates that the IK should be blended in, Lumberyard uses the _IKTarget bone to apply the IK solver listed in the .chrparams file to the bone chain.

The end effector of the bone chain is aligned with the target bone and matching its rotation. In this way, you can also control hand orientation.

Blend weight is determined by the distance (in centimeters) of the _IKBlend weight bone from its parent along the x-axis. The distance is limited to values from 0 to 100 to avoid potential problems from blending multiple animations that might affect the same blend bones.

For best visual results, animate the character to get the end effector close and use the IK only to fix the deviation instead of doing all movement with the IK bones alone.

To make Lumberyard aware of the new IK bones and link them to a solver, open the .chrparams file and add a new line for each to the <Animation_Driven_IK_Targets> section, which lists every bone-controlled IK setup the character uses, as shown in the following example:

```
<Animation_Driven_IK_Targets>
  <ADIKTarget Handle="LftArm01" Target="Bip01 Chin_IKTarget" Weight="Bip01 Chin_IKBlend"/>
</Animation_Driven_IK_Targets>
```

Each entry to the <Animation_Driven_IK_Targets> section specifies which bones to use for the target and the blend weight and includes a handle that points to an IK solver. These handles are listed in the <LimbIK_Definition> section of the .chrparams file, which links a solver and a bone chain.

**Note**
You cannot retarget animations between different skeletons.
Bones without rotation controllers are ignored for optimization purposes.

## Foot IK and Ground Alignment

Lumberyard can automatically adjust a character's legs and feet to match the surface of the terrain the character is walking on. This adjustment includes foot alignment to the direction of the slope, in addition to adjusting the legs to different ground heights.

Leg and foot IK setup is defined in the character .chrparams file. Both legs must be added to the file as follows:

```
<LimbIK_Definition>
  <IK EndEffector="Right_Foot" Handle="RgtLeg01" Root="Right_Thigh" Solver="2BIK"/>
  <IK EndEffector="Left_Foot" Handle="LftLeg01" Root="Left_Thigh" Solver="2BIK"/>
</LimbIK_Definition>
```

The Handle name for the right and left legs must be "RgtLeg01" and "LftLeg01" respectively. You can use any naming for the calf, foot, and thigh as long as they are defined in the .chrparams file. For more information, see Chrparams File Elements (p. 68).
The bones listed following must be named as shown in the list and are required for ground alignment. The last four bones listed are all children of the foot bone.

- **Bip01 pelvis** – The character’s hip joint.
- **Bip01 planeWeightLeft** – For 3ds Max, this bone shares the same x and y position but is approximately 100 cm. above the foot on the z-axis. For Maya, this bone shares the same x and z position but is approximately 100 cm. above the foot on the y-axis.
- **Bip01 planeTargetLeft** – For 3ds Max, this bone shares the same x and y position and is aligned to 0 on the z-axis. For Maya, this bone shares the same x and z position and is aligned to 0 on the y-axis.
- **Bip01 planeWeightRight** – For 3ds Max, this bone shares the same x and y position but is approximately 100 cm. above the foot on the z-axis. For Maya, this bone shares the same x and z position but is approximately 100 cm. above the foot on the y-axis.
- **Bip01 planeTargetRight** – For 3ds Max, this bone shares the same x and y position and is aligned to 0 on the z-axis. For Maya, this bone shares the same x and z position and is aligned to 0 on the y-axis.

The PlaneTarget and PlaneWeight bones are set up to give an absolute offset limit. The aligned pose drives the PlaneTarget node to align to the PlaneWeight node and no further.

**Debugging Ground Alignment Poses**

You can use the following console variables for debugging:

- `a_poseAlignerEnable 1` – Enables alignment.
- `a_poseAlignerDebugDraw 1` – Enables debug drawing of plane weight, target, and root offsets.
- `a_poseAlignerForceWeightOne 1` – Forces the weight to 1, which causes the limb to automatically adjust.

**Limb IK**

You can set up limb IK chains for characters. When a limb IK chain is active, Lumberyard calculates values for the joints that are part of the chain so that the end effector reaches the specified target position.

The behavior for each chain and the number of joints supported depends on the IK solver used: 2BIK for two-bone IK, 3BIK for three-bone IK, and CCDX for cyclic coordinate descent with x joints.

Systems that use limb IK chains include animation-driven IK, foot and leg ground alignment, and game code.

The following summarizes the attributes that you must define for each IK element:

- **EndEffector** – The joint that reaches the target location.
- **Handle** – The limb IK definition. No more than 8 characters are allowed, and the handle must be unique.
- **Root** – The starting joint for the IK chain.
- **Solver** – Code that calculates the joint values.

**Note**

The joint names referenced for the attributes should match the names of the joints in your skeleton, but these names don’t have any specific naming requirements.

The limb IK parameters are stored in the `.chrparams` file with the following format:
Animation Events

Animations in Lumberyard can be marked up to send custom events at a specific time in an animation. This markup is used for time-aligned blending; for example, to match footplants in animations. Another application of animation events is to spawn particle effects at the right moment.

These events can also be used by a variety of systems that need to receive information about when an animation has reached a certain point, such as in combination with a melee system.

Marking Up Animations with Events

Events for animations are stored in an XML file that is loaded when the character starts up. For this to happen automatically, the database must be included in the chrparams file.

Receiving Animation Events in the Game Code

Animation events are passed on to the game object once they have been triggered. The Actor and Player implementations both handle these animation events. See either Actor.cpp or Player.cpp for the function:

```c
void AnimationEvent(ICharacterInstance *pCharacter, const AnimEventInstance &event)
```

Limb IK Technical

Lumberyard's animation system allows the setup of IK chains for characters.

When an IK chain is active, the system calculates the joint angles in the chain so that the end effector (typically a hand or foot) reaches the target position.

Setting Up

IK chains are defined in the chrparams file.

Using LimbIK from Code

To activate a Limb IK chain from outside the Animation system, use the function SetHumanLimbIK, accessible through the ISkeletonPose interface. The SetHumanLimbIK function needs to be called in
each frame in which you want the IK chain to be active. The name of the Limb IK chain is defined in the chrparams file:

```cpp
ISkeletonPose& skeletonPose = ...;
skeletonPose.SetHumanLimbIK(targetPositionWorldSpace, "RgtArm01");
```

## Animation Streaming

Animation is very memory-intensive and tends to use a large amount of resources. Limited memory budgets, high numbers of animated joints, and requirements for high animation quality make it wasteful for a project to keep all animations constantly loaded in memory.

Lumberyard's animation system alleviates this issue by streaming in animation resources (file granularity level) when needed, and unloading them when not needed. Streaming of asset files is achieved by using the DGLINK Streaming System. Streaming assets in and out allows the system to keep only the needed resources in memory—which is done at the expense of complexity, as you must now plan how and when animation resources are used.

### Animation Data

Animation data usage is divided into two main sections:

- The **header** section contains generic information for an animation (filename, duration, flags, etc).
- The **controller** section contains the animation curves. For each joint involved, this section contains information on all the position and orientation values that the joint needs in order to play that animation. Even when compressed, controller data can easily take up more than 95% of the total memory required for an animation.

#### Animation Header Data

Header data for animations is stored in CAF files and in the `animations.img` file.

**CAF** files contain the header information on a single animation, while `animations.img` contains header information for all animations in the build. The `animations.img` is obtained as a result of processing all the animations with the Resource Compiler.

The engine usually loads all the animation files' headers from the `animations.img` file instead of loading from individual files (reading the information from individual files can considerably slow down loading time).

Because of the extreme size difference between controllers and headers, Lumberyard streams only the controller data in and out of memory. The header data for all animations is kept at all times in memory, as it is practical to have that information available at all times.

**Note**

During development—for example, when working with local animation files—you must disable usage of `animations.img` and load the header information from individual CAF files instead. To do so, set the `ca_UseIMG_CAF` console variable to 0 before the engine starts.

#### Animation Controller Data

The controller data for animations is stored in CAF files or DBA files.
• **CAF** files contain controller information for a single animation.
• **DBA** files contain controller information for a group of animations.

When a DBA is loaded, controllers for all animations contained in that DBA are available until the DBA is unloaded. For this reason, it is useful to group animations that are used together in the same DBA. An extra benefit of putting similar animations together in a DBA is that equal controllers are only stored once. This reduces the memory usage of your animations.

### Loading Controller Data

The animation system properly plays animations only when their controllers are in memory.

If controller data is not available when playback of an asset is requested, the animation system streams it in from disk. Streaming of controller data is performed asynchronously—the animation system does not wait until after asset playback is requested. This prevents stalling the system.

If high level systems fail to notify the animation system that they require controller data (see the preload functions section), the animation system does not know that an asset is required until it is requested to play. This is dangerously close to when the controller data is needed. If the controller data is not available in time, it typically leads to visual glitches, which can sometimes be observed, for example, only the first time an animation is played.

Therefore, it is important to have controller data streamed in before playback of an animation is requested. This minimizes undesired glitches that occur while waiting for animation streaming to end.

The amount of time required for streaming to complete depends on many factors, such as the current system load, streaming speed of the target system, size of the resource that needs to be loaded, and so on.

### Unloading Controller Data

The animation system will not unload controller data that is currently in use.

It is possible to prevent unloading of animation data entirely by setting `ca_DisableAnimationUnloading` to 1.

Controllers in **CAF** files are unloaded after the system detects that they are no longer in use. To prevent controllers in **CAF** files from being unloaded, set `ca_UnloadAnimationCAF` to 0.

Controllers in **DBA** files remain in memory until a certain amount of time passes after the animations in them are used. However, if the DBA is locked, controllers are not unloaded until the lock status is set back to 0.

To change the time that the animation system waits to unload controllers in **DBA** files, use the following console variables:

- `ca_DBAUnloadUnregisterTime` – Timeout in seconds after the last usage of a controller and all animations using that DBA; when this timeout is reached, the DBA marks their controller data as 'unloaded'.
- `ca_DBAUnloadRemoveTime` – Timeout in seconds after the last usage of a controller in a DBA; when this timeout is reached, the DBA performs an actual unload from memory. This value should be greater than or equal to `ca_DBAUnloadUnregisterTime`.

The following section describes how to lock individual resources in memory to prevent the system from unloading them.
Preloading and Keeping Controllers in Memory

Preload functions are performed by high level systems or user code (usually game code), as these contain most of the information on when and how assets are accessed. For example, the Track View editor looks a number of seconds ahead in the timeline for any animations that appear, and calls the preload functions.

Preloading Controllers in DBA files

To preload and trigger the streaming of a DBA file:

```cpp
gEnv->pCharacterManager->DBA_PreLoad(dbaFilename, priority);
```

To trigger the streaming of a DBA file, and request a change to the locked state (which specifies whether it should be locked in memory):

```cpp
gEnv->pCharacterManager->DBA_LockStatus(dbaFilename, lockStatus, priority);
```

To unload all controller data in a DBA from memory (unloads data only if none of the controllers are currently being used):

```cpp
gEnv->pCharacterManager->DBA_Unload(dbaFilename);
```

**Note**

To make the system automatically load and lock a DBA file while a character is loaded, use the flags="persistent" in the chrparams file.

Preloading Controllers in CAF files

To increase the reference count of a CAF file:

```cpp
gEnv->pCharacterManager->CAF_AddRef(lowercaseAnimationPathCRC);
```

Controllers for a CAF file start streaming in when its reference count goes from 0 to 1.

To decrease the reference count of a CAF file:

```cpp
gEnv->pCharacterManager->CAF_Release(lowercaseAnimationPathCRC);
```

Controllers for a CAF file are unloaded by the animation system only after the reference count reaches 0 (the animation system, when playing a CAF file, also increases this reference count, so that an animation is not unloaded while in use).

To check whether the controllers for a CAF file are loaded:

```cpp
gEnv->pCharacterManager->CAF_IsLoaded(lowercaseAnimationPathCRC);
```

To synchronously load the controllers for a CAF file:

```cpp
gEnv->pCharacterManager->CAF_LoadSynchronously(lowercaseAnimationPathCRC);
```

Synchronously loading CAF assets is strongly discouraged unless absolutely necessary, as it will likely result in stalls.
Animation Debugging

Several tools are available for debugging animation issues.

Layered Transition Queue Debugging

You can enable on-screen debug information to see which animations are queued and playing, as well as information about the applied pose modifiers and IK.

**Show Per Entity**

To show the transition queue for all the character instances of a specified entity:

```
es_debuganim <entityname> [0 | 1]
```

* `<entityname>`
  * Name of the entity to debug. In a single player game, the player is typically called "dude." Note that the GameSDK example player has both a first person and a third person character instance.

* `[0 | 1]`
  * Specify 1 or no second parameter to turn it on for this specific entity. Specify 0 to turn it off.

**Examples**

To turn on debugging for a player with the entity name "dude":

```
es_debuganim dude 1
```

To turn off debugging for an entity called "npc_flanker_01":

```
es_debuganim npc_flanker_01 0
```

**Show Per CharacterInstance**

You can show the transition queue for all character instances or the ones that have a specific model name.

```
ca_debugtext [<modelname-substring> | 1 | 0]
```

* `<modelname-substring>`
  * Shows information for all character instances whose modelname contains the specified string.

* `[0 | 1]`
  * If 1 is specified, all character instances are shown. If 0 is specified, the debug text is turned off.

**Examples**

To show information on all character instances with "player" in their model name:
ca_debugtext player

To turn off all transition queue information:

ca_debugtext 0

Interpreting the Output

Each animation in the transition queue is displayed as in the following example. Key elements of this display are described following the example.

AnimInAFIFO 02: t:1043 _stand_tac_idle_scar_3p_01 ATime:0.84 (1.17s/1.40s) ASpd:1.00
Flag:00000042 (--------I-K----) TTime:0.20 TWght:1.00 seg:00 inmem:1
(Try)UseAimIK: 1 AimIKBlend: 1.00 AimIKInfluence: 1.00 (Try)UseLookIK: 0 LookIKBlend: 0.00
LookIKInfluence: 0.00
MoveSpeed: 4.49 locked: 1
PM class: AnimationPoseModifier_OperatorQueue, name: Unknown
... LayerBlendWeight: 1.00 ...
... ADIK Bip01 RHand2RiflePos_IKTarget: 0.24 Bip01 RHand2Aim_IKTarget: 1.00 Bip01
LHand2Aim_IKTarget: 0.00

Text Color

• When an animation is not yet active, it is in black or green.
• When an animation is active, it is in red or yellow.

Or in detail:

• Red Channel = Animation Weight
• Green Channel = (layerIndex > 0)
• Alpha Channel = (Weight + 1)*0.5

AnimInAFIFO Line (one per animation)

AnimInAFIFO 02: t:1043 _stand_tac_idle_scar_3p_01 ATime:0.84 (1.17s/1.40s) ASpd:1.00
Flag:00000042 (--------I-K----) TTime:0.20 TWght:1.00 seg:00 inmem:1

AnimInAFIFO 02

Layer index (decimal, zero-based)
t:1043

User token (decimal)
_stand_tac_idle_scar_3p_01

Animation name (alias) of the currently playing animation, aim/look-pose or bspace
ATime:0.84 (1.17s/1.40s)

ATime:XXX (YYYYs/ZZZZs)
• XXXX = Current time in 'normalized time' (0.0...1.0) within the current segment
• YYYY = Current time (seconds) within the current segment
• ZZZZ = Expected duration (seconds) of the current segment

ASpd: 1.00

Current animation speed (1.0 = normal speed)

Flag: 00000042 (--I-K--)

Animation Flags

Flag: XX555555 (+ybFx3nSIAKTRLM)

The first number is the animation flags in hexadecimal

Between parentheses you see the individual flags:

<table>
<thead>
<tr>
<th>char</th>
<th>flag</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>CA_FORCE_TRANSITION_TO_ANIM</td>
<td>0x008000</td>
</tr>
<tr>
<td>y</td>
<td>CA_FULL_ROOT_PRIORITY</td>
<td>0x004000</td>
</tr>
<tr>
<td>b</td>
<td>CA_REMOVE_FROM_FIFO</td>
<td>0x002000</td>
</tr>
<tr>
<td>V</td>
<td>CA_TRACK_VIEW_EXCLUSIVE</td>
<td>0x001000</td>
</tr>
<tr>
<td>F</td>
<td>CA_FORCE_SKELETON_UPDATE</td>
<td>0x000800</td>
</tr>
<tr>
<td>x</td>
<td>CA_DISABLE_MULTILAYER</td>
<td>0x000400</td>
</tr>
<tr>
<td>3</td>
<td>CA_KEYFRAME_SAMPLE_30Hz</td>
<td>0x000200</td>
</tr>
<tr>
<td>n</td>
<td>CA_ALLOW_ANIM_RESTART</td>
<td>0x000100</td>
</tr>
<tr>
<td>S</td>
<td>CA_MOVE2IDLE</td>
<td>0x000080</td>
</tr>
<tr>
<td>I</td>
<td>CA_IDLE2MOVE</td>
<td>0x000040</td>
</tr>
<tr>
<td>A</td>
<td>CA_START_AFTER</td>
<td>0x000020</td>
</tr>
<tr>
<td>K</td>
<td>CA_START_AT_KEYTIME</td>
<td>0x000010</td>
</tr>
<tr>
<td>T</td>
<td>CA_TRANSITION_TIMEWARPING</td>
<td>0x000008</td>
</tr>
<tr>
<td>R</td>
<td>CA_REPEAT_LAST_KEY</td>
<td>0x000004</td>
</tr>
<tr>
<td>L</td>
<td>CA_LOOP_ANIMATION</td>
<td>0x000002</td>
</tr>
<tr>
<td>M</td>
<td>CA_MANUAL_UPDATE</td>
<td>0x000001</td>
</tr>
</tbody>
</table>

TTime: 0.20

Transition Time

Total length of transition into this animation in seconds (this is static after pushing the animation)

TWght: 1.00

Transition Weight

Current weight of this animation within the transition (0 = not faded in yet, 1 = fully faded in)

seg: 00

Current segment index (zero-based)
inmem: 1

Whether or not the animation is in memory (0 basically means it's not streamed in yet)

**Aim/Look-IK Line**

<table>
<thead>
<tr>
<th>(Try)UseAimIK: 1</th>
<th>AimIKBlend: 1.00</th>
<th>AimIKInfluence: 1.00</th>
<th>(Try)UseLookIK: 0</th>
<th>LookIKBlend: 0.00</th>
<th>LookIKInfluence: 0.00</th>
</tr>
</thead>
</table>

**(Try)UseAimIK: 1**

Whether Aim IK is turned on or not (set using PoseBlenderAim::SetState)

*AimIKBlend: 1.00*

Weight value requested for Aim IK (could go up and down based on fade times, etc.)

*AimIKInfluence: 1.00*

Final influence weight value of AimIK (\(\text{smoothed} (\text{clamped}(\text{AimIKBlend})) \times \text{weightOfAllAimPoses}\))

**(Try)UseLookIK: 0**

Whether Look IK is turned on or not

*LookIKBlend: 0.00*

Weight value requested for Look IK (could go up and down based on fade times, etc.)

*LookIKInfluence: 0.00*

Final influence weight value of LookIK (\(\text{smoothed} (\text{clamped}(\text{LookIKBlend})) \times \text{weightOfAllLookPoses}\))

**Parameter Line(s) (only for blend spaces)**

<table>
<thead>
<tr>
<th>MoveSpeed: 4.500000</th>
<th>locked: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>TravelAngle: 0.000000</td>
<td>locked: 0</td>
</tr>
</tbody>
</table>

*MoveSpeed: 4.500000*

Value for the specified blend space parameter (MoveSpeed in this case)

*locked: 1*

Whether or not the parameter is locked (= unable to change after it is set for the first time)

**PoseModifier Lines (if running)**

*PM class: AnimationPoseModifier_OperatorQueue, name: Unknown*

Displays which pose modifiers are running in this layer. Shows the class as well as the name (if available).

**LayerBlendWeight Line (not on layer 0)**

<table>
<thead>
<tr>
<th>LayerBlendWeight: 1.00</th>
</tr>
</thead>
</table>

The weight of this layer (0.00 - 1.00)
ADIK Line(s) (only if animation driven IK is applied)

| ADIK Bip01 RHand2RiflePos_IKTarget: 0.24 Bip01 RHAnd2Aim_IKTarget: 1.00 Bip01 LHand2Aim_IKTarget: 0.00 |

Displays a list of the animation driven IK targets and their current weight. For more detailed position/rotation information, use the separate console variable `ca_debugadiktargets 1`.

CommandBuffer Debugging

At the lowest level, the animation system executes a list of simple commands to construct the final skeleton's pose.

These commands are, for example, "sample animation x at time t, and add the result with weight w to the pose". Or "clear the pose".

To enable on-screen debug information to see what is pushed on the command buffer (for all characters), use the following command:

```
ca_debugcommandbuffer [0 | 1]
```

Warning Level

To control when the animation system produces warnings using the `ca_animWarningLevel` console variable:

```
ca_animWarningLevel [0 | 1 | 2 | 3]
```

0

Non-fatal warnings are off.

1

Warn about illegal requests.

For example, requesting to start animations with an invalid index.

2

Also warn about things like 'performance issues.'

For example, animation-queue filling up. This might 'spam' your console with a dump of the animation queue at the time of the issue.

3 (default)

All warnings are on. This includes the least important warnings; for example, a warning when playing uncompressed animation data.

Fall and Play

"Fall and Play" activates when a character is ragdollized (on an interface level, it is called `RelinquishCharacterPhysics`) with a >0 stiffness. This activates angular springs in the physical ragdoll that attempts to bring the joints to the angles specified in the current animation frame. The character also tries to select an animation internally based on the current fall and play stage. If there are
none, or very few, physical contacts, this will be a falling animation; otherwise it will be the first frame of a standup animation that corresponds to the current body orientation.

Standup is initiated from outside the animation system through the fall and play function. During the standup, the character physics is switched back into an alive mode and his final physical pose is blended into a corresponding standup animation. This, again, is selected from a standup anims list to best match this pose.

Filename convention for standup animations: When an animation name starts with "standup", it is registered as a standup animation. Also, a type system exists which categorizes standup animations by the string between "standup_" and some keywords ("back", "stomach", "side"). You can control which type to use with CSkeletonPose::SetFnPAnimGroup() methods. At runtime, the engine checks the most similar standup animation registered to the current lying pose and blends to it.

Some example filenames:
- standUp_toCombat_nw_back_01
- standUp_toCombat_nw_stomach_01

While the character is still a ragdoll, it is also possible to turn off the stiffness with a GoLimp method.

Time in the Animation System

The Animation system uses different units of ‘time,’ depending on the system. How those units of time compare is best explained using an example.

The definition of ‘frames’: The Animation system uses a fixed rate of 30 frames per second (fps). Of course, games can run at higher frame rates, but some operations in the Editor that use the concept of ‘frames’—or operations that clamp the animation duration to ‘one frame’—assume a frame rate of 30 fps.

Assume then that you have an animation with a duration of 1.5 seconds. This means that the animation has 46 frames (note that this includes the final frame). So, in the case of Real Time, assume an animation starts at time 0, has no segmentation, and is played back at normal speed. However, rather than using Real Time, the Animation system typically uses Animation Normalized Time. This is compared with Real Time in the following table:

<table>
<thead>
<tr>
<th>Frame Index</th>
<th>Real Time (seconds)*</th>
<th>Animation Normalized Time**</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0 s</td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>0.033.. s = 1/30 s</td>
<td>0.022.. = 1/45</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
<tr>
<td>30</td>
<td>1.0 s</td>
<td>0.666.. = 30/45</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
<tr>
<td>44</td>
<td>1.466.. s = 44/30 s</td>
<td>0.977.. = 44/45</td>
</tr>
<tr>
<td>45</td>
<td>1.5 s = 45/30 s</td>
<td>1.0</td>
</tr>
</tbody>
</table>

* Real time is used to define duration:
- Duration = lastFrame.realTime - firstFrame.realTime. That's 1.5s in our example.
• IAnimationSet::GetDuration_sec() returns the duration of an animation.

**Note:** For a parametric animation, this returns only a crude approximation—the average duration of all its examples, ignoring parameters or speed scaling.

• CAnimation::GetExpectedTotalDurationSeconds() returns the duration of an animation that is currently playing back.

**Note:** For a parametric animation, this returns only a crude approximation, assuming the parameters are the ones that are currently set and never change throughout the animation.

• No function exists that returns the Real Time of an animation. To calculate that, you must manually multiply Animation Normalized Time with the duration.

**Animation Normalized Time:**

• Time relative to the total length of the animation.

• Starts at 0 at the beginning of the animation and ends at 1 (= RealTime/Duration = Keytime/LastKeyTime).

• Used by functions such as ISkeletonAnim::GetAnimationNormalizedTime() and ISkeletonAnim::SetAnimationNormalizedTime().

• Is not well-defined for parametric animations with examples that have differing numbers of segments. For more information, see the following section, Segmentation.

**Segmentation**

In practice, the animation system does not use Animation Normalized Time; this terminology was used to make the introduction easier to understand. Typically, Segment Normalized Time is used. To understand Segment Normalized Time, you must first understand segmentation.

For time warping (phase matching) purposes, animations can be split into multiple segments. For example, to time warp from a walk animation with 2 cycles to a walk animation with 1 cycle, you have to annotate the first animation and split it into two (these are segments). To achieve this segmentation, you must add a segment1 animation event at the border between the cycles.

**Note**

An animation without segmentation has exactly 1 segment, which runs from beginning to end.

Segmentation introduces a new unit for time, Segment Normalized Time, which is time relative to the current segment duration.

Extending our example further, observe what happens when a segment1 animation event at 1.0s is added to split the animation into two segments.

<table>
<thead>
<tr>
<th>Frame Index</th>
<th>Real Time</th>
<th>AnimEvents</th>
<th>(Animation) Normalized Time</th>
<th>Segment Index*</th>
<th>Segment Normalized Time**</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0 s</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>0.033.. s</td>
<td>0.022..</td>
<td>0</td>
<td>0</td>
<td>0.033.. = 1/30</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
<tr>
<td>30</td>
<td>1.0 s</td>
<td>segment1</td>
<td>0.666..</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>
Lumberyard Legacy Reference

Playback Speed

<table>
<thead>
<tr>
<th>Frame Index</th>
<th>Real Time</th>
<th>AnimEvents</th>
<th>(Animation) Normalized Time</th>
<th>Segment Index*</th>
<th>Segment Normalized Time**</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td>1.466.. s</td>
<td>0.977..</td>
<td></td>
<td>1</td>
<td>0.933.. = 14/15</td>
</tr>
<tr>
<td>45</td>
<td>1.5 s</td>
<td>1.0</td>
<td></td>
<td>1</td>
<td>1.0</td>
</tr>
</tbody>
</table>

* Segment index:
- Identifies which segment you are currently in. Runs from 0 to the total number of segments minus 1.
- While an animation is playing, you can use CAnimation::GetCurrentSegmentIndex() to retrieve it.
- When using ca_debugtext or es_debuganim, then this index is displayed after "seg:"

** Segment normalized time:
- Time relative to the current segment's duration.
- 0 at the beginning of the segment, 1 at the end (only 1 for the last segment, as you can see in the table).
- While an animation is playing, you can use CAnimation::Get/SetCurrentSegmentNormalizedTime() to get or set the Segment Normalized Time.
- As the names suggest, CAnimation::GetCurrentSegmentIndex() retrieves the current segment index and CAnimation::GetCurrentSegmentExpectedDurationSeconds() retrieves the duration of the current segment.
- When representing time within parametric animations, it is more convenient to use Segment Normalized Time than Animation Normalized Time; therefore, Segment Normalized Time is used at runtime.
- AnimEvent time is specified using Animation Normalized Time (except for the special case of parametric animation; see the following section).
- When using ca_debugtext or es_debuganim, Segment Normalized Time is displayed after "ATime:"
- Following that, the real time within the segment and the segment duration are displayed within the parentheses.

Playback Speed

Playback speed does not impact the functions that compute duration of playing animations, such as CAnimation::GetExpectedTotalDurationSeconds() or ISkeletonAnim::CalculateCompleteBlendSpaceDuration().

Segmented Parametric Animation

Animation Normalized Time, Segment Index, and Duration all create ambiguity for segmented parametric animations. This is because each example animation within the parametric animation can have its own number of segments. To avoid ambiguity, animation events in or on segmented parametric animations use Segment Normalized Time. As a result, an animation event will be fired multiple times (once per segment) during the animation.

- ISkeletonAnim::GetAnimationNormalizedTime() uses a heuristic: It currently looks for the example animation with the largest number of segments and returns the animation normalized time within that example.
• ISkeletonAnim::GetCurrentSegmentIndex() uses a different heuristic: It currently returns the segment index in the example animation, which happens to be the first in the list.

Given this, we are considering redefining the above based on the following observation: You can define the total number of segments in a parametric animation as the number of segments until repetition starts.

So, say you have a parametric animation consisting of 2 examples—one with 2 segments and the other with 3 segments. This will start to repeat after 6 segments (the lowest common multiple of 2 and 3). However, you can uniquely identify each possible combination of segments using any number from 0 to 5.

The Character Tool uses this method to achieve a well-defined duration. The ISkeletonAnim::CalculateCompleteBlendSpaceDuration() function calculates the duration until the parametric animation starts to repeat (assuming the parameters remain fixed). It reverts to the regular GetExpectedTotalDurationSeconds() implementation for non-parametric animations so that the function can be used in more general situations.

**Animation with Only One Key**

Normally your animations have at least two keys. However, when you convert these into additive animations, the first frame is interpreted as the base from which to calculate the additive, leaving only 1 frame in the additive animation (this means that, in respect to the asset, both the start and end time of the asset are set to 1/30 s).

Functions retrieving the total duration of this animation will return 0.0 (for example, IAnimationSet::GetDuration_sec(), ISkeletonAnim::CalculateCompleteBlendSpaceDuration(), and CAnimation::GetExpectedTotalDurationSeconds()).

However, for playback purposes, the animation system handles these animations as if they have a duration of 1/30th of a second. For example, Animation Normalized Time still progresses from 0 to 1, while real time goes from 0 to 1/30th of a second. CAnimation::GetCurrentSegmentExpectedDurationSecondsx() also returns 1/30th of a second in this case.

**Direction of Time**

Time typically cannot run backward when playing an animation. You can move time backward only if you do it manually by setting the flag CA_MANUAL_UPDATE on the animation and using CAnimation::SetCurrentSegmentNormalizedTime. See the example DGLINK CProceduralClipManualUpdateList::UpdateLayerTimes().

**Time within Controllers**

Different units are used for controllers that contain the actual key data and are used for animation sampling.

<table>
<thead>
<tr>
<th>Frame Index</th>
<th>Real Time</th>
<th>_CAF Ticks*</th>
<th>Keytime**</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0 s</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>0.033.. s</td>
<td>160</td>
<td>1.0</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>
**Frame Index** | **Real Time** | **I_CAF Ticks*** | **Keytime****
---|---|---|---
30 | 1.0 s | 4800 | 30.0
.. | .. | .. | ..
44 | 1.466.. s | 7040 | 44.0
45 | 1.5 s | 7200 | 45.0

* I_CAF Ticks:
  - Used within I_CAF files to represent time
  - There are 4800 I_CAF ticks per second (this is currently expressed by the fact that TICKS_CONVERT = 160 in Controller.h, which assumes 30 keys/second)

** Keytime
  - Used at runtime to pass time to the controllers for sampling animation
  - Used within CAF files to represent time
  - A floating point version of 'frame index'
  - Can represent time in between frames
  - Use GlobalAnimationHeaderCAF::NTime2KTime() to convert from Animation Normalized Time to Keytime
  - All animation controllers in the runtime use Keytime

Animation assets can also have a StartTime other than 0.0s—this complicates matters slightly, but only for the controllers. Typically, for everywhere but the controllers, time is taken relative to this StartTime.
Cloud 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 about the latest features, see the Amazon Lumberyard User Guide.

You can create realistic-looking clouds in your level that move, cast shadows, and that objects can fly through. To use cloud effects, you must enable the Sky Clouds gem.

Note
For information about creating clouds with the component entity system, see Adding Clouds in the Amazon Lumberyard User Guide.

Topics
- Placing Simple Clouds (p. 162)
- Placing Complex Clouds (p. 163)
- Setting Cloud Shading Parameters (p. 164)
- Adding Distance (2D) Clouds (p. 164)
- Using 3D Clouds (p. 165)
- Creating 3D Cloud Templates (p. 166)
- Using Cloud Volumes (p. 167)
- Using Skydomes (p. 168)

Placing Simple Clouds

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 Amazon Lumberyard Legacy Reference.

You can place simple clouds with sprite-based shading and customize it for your level by choosing your cloud texture and modifying such properties as movement speed, size, movement from wind, and so on.

To add simple clouds to your level
1. In the Rollup Bar’s Objects tab, click Entity.
2. Under Browser, expand Render.
3. Drag the Cloud entity into your scene.

Files Associated with Simple Clouds

The following are files associated with simple clouds.
### Configuring Simple Clouds

You can configure the properties for your simple clouds under **Entity Params** and **Entity Properties**.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloudFile</td>
<td>The <code>.xml</code> file containing the description of the cloud</td>
</tr>
<tr>
<td>Scale</td>
<td>The scale of the cloud.</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>This feature will be unavailable in a future release.</td>
</tr>
<tr>
<td>Movement</td>
<td></td>
</tr>
<tr>
<td>AutoMove</td>
<td>Enables cloud movement</td>
</tr>
<tr>
<td>FadeDistance</td>
<td>The distance in meters at which the cloud fades in when moving from one side of the space loop box to the other.</td>
</tr>
<tr>
<td>SpaceLoopBox</td>
<td>The size of the box in which the volume object moves from one end to the other.</td>
</tr>
<tr>
<td>Speed</td>
<td>The rate of movement in the x, y, and z dimensions</td>
</tr>
</tbody>
</table>

### Placing Complex Clouds

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 Amazon Lumberyard Legacy Reference.

You can place more complex clouds, also called volume objects, which feature complex voxelized three-dimensional volume shading.

**To add complex clouds to your level**

1. In the Rollup Bar's **Object** tab, click **Entity**.
2. Under **Browser**, expand **Render**.
3. Drag the **VolumeObject** entity into your scene.

### Files Associated with Complex Clouds

The following are files associated with volume objects, or complex clouds.
### Configuring Complex Clouds

You can configure the properties for your complex clouds under **Entity Params** and **Entity Properties**.

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VolumeObjectFile</td>
<td>The <code>.xml</code> file containing the description of the cloud</td>
</tr>
<tr>
<td>Movement</td>
<td></td>
</tr>
<tr>
<td>AutoMove</td>
<td>Enables volume object movement</td>
</tr>
<tr>
<td>FadeDistance</td>
<td>The distance in meters at which the cloud fades in when moving from one side of the space loop box to the other</td>
</tr>
<tr>
<td>SpaceLoopBox</td>
<td>The size of the box in which the volume object moves from one end to the other</td>
</tr>
<tr>
<td>Speed</td>
<td>The rate of movement in the x, y, and z dimensions</td>
</tr>
</tbody>
</table>

### Setting Cloud Shading Parameters

Cloud shading, unlike cloud shadows, effects 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, click **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 Distance (2D) Clouds

Distance clouds are two-dimensional clouds that are placed at a great distance, such as on the horizon or at high altitudes. They are suitable for creating a realistic distant sky scene without causing rendering performance issues. Distance clouds do not move and cannot be flown through.
Using the Distance Cloud shader, you can create distance clouds with no shading (simple clouds) or with more accurate shading (advanced clouds).

To create a 2D distance cloud
1. In the Rollup Bar, under Objects, click Misc, Distance Cloud.
2. Click Materials/Clouds/DistanceClouds.
3. In Material Editor, under Shader Generation Params, select either the Simple distance clouds or Advanced distance clouds check box.
4. Adjust Opacity and Lighting settings as needed.
5. Click Assign Item to Selected Objects and close Material Editor.
6. Drag the cloud to the desired location in your level, placing it far enough away that a player will not notice it is flat.

Using 3D Clouds

The 3D clouds effect use the Common Cloud shader, which uses gradient lighting for each vertex and takes the sun, cloud, and viewer positions into account. Gradient lighting interpolates between the bright color, which is calculated from the Sun color multiplier, and the dark color, which is calculated from the Sky color multiplier. For more information on sun and sky color parameters, see Adding Sky Effects.

In addition, rim lighting is also applied to each pixel to capture the effects of light scattering when clouds are backlit by the sun.

3D clouds use soft clipping to gradually fade in and out at the clipping planes. This prevents rendering artifacts for distant clouds as well as flickering due to cloud particles entering and leaving the view during a flythrough. Additionally, clouds blend softly against opaque scene geometry.

For best results, place 3D at a height greater than 1000 meters (Z value = 1000).

To create a 3D cloud
1. In the Rollup Bar, under Objects, click Entity.
2. Under Browser, select a suitable asset.
3. Click No Custom Material.
4. Under Shader Params, adjust parameter values as described in Common Cloud shader.
5. Click Assign Item to Selected Objects and close Material Editor.
6. Drag the cloud to the desired location in your level.
7. Under Entity Properties, click CloudFile. Then click the folder icon.
8. In Preview, click to select a cloud file.
9. Click and adjust the values of the following parameters:
   - CloudFile – Sets the cloud material.
   - Scale – Sets the scale factor for sizing the cloud.
   - AutoMove – Select to enable cloud movement. Disable first before selecting a cloud to edit.
   - FadeDistance – Sets the distance from the edges of the SpaceLoopBox that the cloud fades in and out from.
   - SpaceLoopBox – Sets the size of the area that the cloud moves in from one end to the other.
   - Speed – Sets the speed at which the cloud moves. For normal clouds, use a value around 5. For storm clouds, use a value around 15.
Creating 3D Cloud Templates

You can use the Clouds tool in Lumberyard Editor to create new cloud template XML files. You can use those template files later to add and place clouds as described in previous procedures.

The basic process for creating a cloud template is to create an area box that defines the size of the cloud, assign a material, select the Common Cloud shader, and then export and save the template. See the following procedure for details.

All clouds use a texture map, which is made up of multiple sprites that are organized into columns and rows, as the following image shows. You create cloud texture maps using your DCC tool.

To create a new 3D cloud template

1. In the Rollup Bar, under Objects, click Area, AreaBox.
2. Under AreaBox, click <No Custom Material>.
3. In the Material Editor, select the cloud texture map you created in your DCC.
5. Under Shader Params, adjust the parameters for the desired effect.
6. Click Assign item to Selected Objects. Close the Material Editor.
7. Click to place the area box in your level.
8. In Lumberyard Editor, click Game, Clouds, Create, and type a name for the cloud template.
9. Under Cloud Params, adjust the following cloud texture map parameters for desired effect.

   **Number of Rows**
   
   Sets the number of sprite rows in the cloud texture. Leave at 4 when using the default cumulus_01.dds texture.

   **Number of Columns**
   
   Sets the number of sprite columns in the cloud texture. Leave at 4 when using default cumulus_01.dds texture.

   **Sprite Row**
   
   Designates a row in the cloud texture for rendering.

   **Number of Sprites**
   
   Sets the number of sprites to be generated in the cloud.

   **Size of Sprites**
   
   Sets the scale of the sprites in the cloud.
Size Variation

Defines the randomization in size of the sprites within the cloud.

Angle Variations

Defines limits of randomization in the rotation of the sprites within the cloud.

Minimal Distance between Sprites

Defines the minimum distance between the generated sprites within the cloud.

Every Box has Sprites

Specifies that each box has sprites.

Density

Defines the density of the cloud.

Show Particles like Spheres

Turns on additional sphere rendering for each sprite generated.

Preview Cloud

Renders the generated cloud.

Auto Update

Updates the cloud rendering automatically with each parameter change.

10. Click Generate Clouds. The cloud should be visible inside the AreaBox in your level.

11. Click Export, then save the cloud template in a suitable directory.

Using Cloud Volumes

Cloud volumes are 3D volumetric clouds that use the Volume Object shader to achieve very realistic shading effects. Unlike 3D clouds, cloud volumes feature correct self-shadowing. Adding many cloud volumes to a level can deteriorate performance.

To create a cloud volume

1. In the Rollup Bar, under Objects, click Entity.
2. Under Browser, select your asset.
3. Click No Custom Material.
4. Expand the left tree and select your asset.
5. Under Material Settings, for Shader, select VolumeObject.
6. Under Shader Params and Shader Generation Params, adjust parameter values as described in Volume Object shader.
7. Click Assign Item to Selected Objects and close Material Editor.
8. Drag the cloud to the desired location in your level.
9. Under Entity Properties, adjust the following parameters:
   - CloudFile – The cloud material.
   - Scale – The scale factor for sizing the cloud.
   - AutoMove – Select to enable cloud movement.
   - FadeDistance – The distance from the edges of the space loop box from which the cloud fades in and out.
• **SpaceLoopBox** – The size of the volume that the clouds moves in from one end to the other.

• **Speed**: The speed at which cloud moves. For normal clouds, use a value around 5. For storm clouds, use a value around 15.

**Tip**
If you want to change the settings for an existing cloud, disable **AutoMove** first to make it easier to select the cloud in your level.

---

**Using Skydomes**

As the name suggests, skydomes simulate the entire sky, complete with clouds, in a level. They are cheap to render and provide good results. You can also use Flow Graph to animate skydomes.

**To add a skydome**

1. In the Rollup Bar, under **Objects**, click **Geom Entity**.
2. Under **Browser**, select a suitable asset.
3. Drag the object to the desired location in your level.
4. At the bottom of Lumberyard Editor, adjust the x and y values to center the skydome pivot in the level.
5. Scale the skydome larger as needed for more realism.
The Code\CryCommon directory is the central directory for legacy engine interfaces (as well as some commonly used code for reuse).

This section includes the following topics:
- CryExtension (p. 169)
- CryString (p. 192)
- ICrySizer (p. 193)
- Legacy Serialization Library (p. 193)

CryExtension

Note
The use of extensions has been deprecated in Lumberyard. Gems are now the recommended way to add modules to your Lumberyard game. For more information, see Gems.

The complexity of Lumberyard can be challenging to both newcomers and experienced users who want to understand, configure, run, and extend it. Refactoring Lumberyard into extensions makes it easier to manage. Existing features can be unplugged (at least to some degree), replaced, or customized, and new features added. Extensions can consolidate code for a single feature in one location. This avoids having to implement a feature piecemeal across a number of the engine's base modules. Refactoring into extensions can also make the system more understandable at a high level.

Lumberyard's extension framework is loosely based on some fundamental concepts found in Microsoft's Component Object Model (COM). The framework defines two base interfaces that each extension needs to implement, namely ICryUnknown and ICryFactory. These are similar to COM's IUnknown and IClassFactory. The interfaces serve as a base to instantiate extensions, allow interface type casting, and enable query and exposure functionality.

The framework utilizes the concept of shared pointers and is implemented in a way to enforce their consistent usage to help reduce the chance of resource leaks. A set of C++ templates wrapped in a few macros is provided as Glue Code Macros (p. 175) that encourage engine refactoring into extensions. The glue code efficiently implements all base services and registers extensions within the engine. Additionally, a few helper functions implement type-safe casting of interface pointers, querying the IDs of extension interfaces, and convenient instantiation of extension classes. Hence, repetitive writing of tedious boilerplate code is unnecessary, and the potential for introducing bugs is reduced. An example is provided in the section Using Glue Code (p. 183). If the provided glue code is not applicable, then you must implement the interfaces and base services manually, as described in the section Without Using Glue Code (p. 185).

Clients access extensions through a system wide factory registry. The registry allows specific extension classes to be searched by either name or ID, and extensions to be iterated by using an interface ID.

Composites

The framework allows extensions to expose certain internal objects that they aggregate or are composed of. These so called composites are extensions themselves because they inherit from ICryUnknown.
Composites allow you to reuse desired properties like type information at runtime for safe casting and loose coupling.

**Shared and Raw Interface Pointers**

Although the framework was designed and implemented to utilize shared pointers and enforce their usage in order to reduce the possibility of resource leaks, raw interface pointers can still be acquired. Therefore, care needs to be taken to prevent re-wrapping those raw interface pointers in shared pointer objects. If the original shared pointer object is not passed during construction so that its internal reference counter can be referred to, the consistency of reference counting will be broken and crashes can occur. A best practice is to use raw interface pointers only to operate on interfaces temporarily, and not store them for later use.

**GUIDs**

You must use globally unique identifiers (GUIDs) to uniquely identify extensions and their interfaces. GUIDs are essentially 128-bit numbers generated by an algorithm to ensure they only exist once within a system such as Lumberyard. The use of GUIDs is key to implementing the type-safe casting of extension interfaces, which is particularly important in large scale development projects. To create GUIDs, you can use readily available tools like the **Create GUID** feature in Visual Studio or the macro below.

GUIDs are defined as follows.

```c++
struct CryGUID
{
    uint64 hipart;
    uint64 lopart;

    ...
};
typedef CryGUID CryInterfaceID;
typedef CryGUID CryClassID;
```

Declared in the following framework header files:

- `\dev\Code\CryEngine\CryCommon\CryExtension\CryGUID.h`
- `\dev\Code\CryEngine\CryCommon\CryExtension\CryTypeID.h`

The following Visual Studio macro can be used to generate GUIDs conveniently within the IDE. The macro writes GUIDs to the current cursor location in the source code editor window. Once added to **Macro Explorer**, the macro can be bound to a keyboard shortcut or (custom) toolbar.

```vbs
Public Module CryGUIDGenModule

Sub GenerateCryGUID()
    Dim newGuid As System.Guid
    newGuid = System.Guid.NewGuid()

    Dim guidStr As String
    guidStr = newGuid.ToString("N")
    guidStr = guidStr.Insert(16, ", 0x")
    guidStr = guidStr.Insert(0, "0x")

    REM guidStr = guidStr + vbCrLf
    REM guidStr = guidStr + newGuid.ToString("D")
End Sub
```

170
ICryUnknown

ICryUnknown provides the base interface for all extensions. If making it the top of the class hierarchy is not possible or desired (for example, in third party code), you can apply an additional level of indirection to expose the code by using the extension framework. For an example, see If ICryUnknown Cannot Be the Base of the Extension Class (p. 190).

ICryUnknown is declared as follows.

```cpp
struct ICryUnknown
{
    CRYINTERFACE_DECLARE(ICryUnknown, 0x1000000010001000, 0x1000100000000000)

    virtual ICryFactory* GetFactory() const = 0;

protected:
    virtual void* QueryInterface(const CryInterfaceID& iid) const = 0;
    virtual void* QueryComposite(const char* name) const = 0;
};

typedef boost::shared_ptr<ICryUnknown> ICryUnknownPtr;
```

- GetFactory() returns the factory with which the specified extension object was instantiated. Using the provided glue code this function has constant runtime.

- QueryInterface() returns a void pointer to the requested interface if the extension implements it, or NULL otherwise. This function was deliberately declared as protected to enforce usage of type-safe interface casting semantics. For information on casting semantics, see Interface casting semantics (p. 173). When the provided glue code is used, this function has a (worst case) run time that is linear in the number of supported interfaces. Due to glue code implementation details, no additional internal function calls are needed. A generic code generator produces a series of instructions that compares interface IDs and returns a properly cast pointer.

- QueryComposite() returns a void pointer to the queried composite if the extension exposes it; otherwise, NULL. As with QueryInterface(), this function was deliberately declared as protected to enforce type querying. For information on type querying, see Querying composites (p. 174). The function has a (worst case) run time linear in the number of exposed composites.

- Unlike in COM, ICryUnknown does not have AddRef() and Release(). Reference counting is implemented in an non-intrusive way by using shared pointers that are returned by the framework when extension classes are instantiated.

Declared in the following framework header file:

- CryCommon/CryExtension/ICryUnknown.h

ICryFactory

ICryFactory provides the base interface to instantiate extensions. It is declared as follows.
struct ICryFactory
{
    virtual const char* GetClassName() const = 0;
    virtual const CryClassID& GetClassID() const = 0;
    virtual bool ClassSupports(const CryInterfaceID& iid) const = 0;
    virtual void ClassSupports(const CryInterfaceID*& pIIDs, size_t& numIIDs) const = 0;
    virtual ICryUnknownPtr CreateClassInstance() const = 0;
    protected:
    virtual ~ICryFactory() {};
};

- GetClassName() returns the name of the extension class. This function has constant run time when the provided glue code is used.

- GetClassID() returns the ID of the extension class. This function has constant run time when the provided glue code is used.

- ClassSupports(iid) returns true if the interface with the specified ID is supported by the extension class; otherwise, false. This function has a (worst case) run time linear in the number of supported interfaces when the provided glue code is used.

- ClassSupports(pIIDs, numIIDs) returns the pointer to an internal array of IDs enumerating all of the interfaces that this extension class supports as well as the length of the array. This function has constant run time when the provided glue code is used.

- CreateClassInstance() dynamically creates an instance of the extension class and returns a shared pointer to it. If the extension class is implemented as a singleton, it will return a (static) shared pointer that wraps the single instance of that extension class. This function has constant run time when the provided glue code is used, except for the cost of the constructor call for non-singleton extensions.

- The destructor is declared protected to prevent explicit destruction from the client side by using delete, boost::shared_ptr<T>, etc. ICryFactory instances exist (as singletons) throughout the entire lifetime of any Lumberyard process and must not be destroyed.

Declared in the following framework header file:

- CryCommon/CryExtension/ICryFactory.h

ICryFactoryRegistry

ICryFactoryRegistry is a system-implemented interface that enables clients to query extensions. It is declared as follows.

struct ICryFactoryRegistry
{
    virtual ICryFactory* GetFactory(const char* cname) const = 0;
    virtual ICryFactory* GetFactory(const CryClassID& cid) const = 0;
    virtual void IterateFactories(const CryInterfaceID& iid, ICryFactory** pFactories, size_t& numFactories) const = 0;
    protected:
};
virtual -ICryFactoryRegistry() {};

- GetFactory(cname) returns the factory of the extension class with the specified name; otherwise, NULL.

- GetFactory(cid) returns the factory of the extension class with the specified ID; otherwise, NULL.

- IterateFactory() if pFactories is not NULL, IterateFactory copies up to numFactories entries of pointers to extension factories that support iid. numFactories returns the number of pointers copied. If pFactories is NULL, numFactories returns the total amount of extension factories that support iid.

- The destructor was declared protected to prevent explicit destruction from the client side by using delete, boost::shared_ptr<T>, etc. ICryFactoryRegistry is a system interface and that exists throughout the entire lifetime of any CryEngine process and must not be destroyed.

Declared in the following framework header file:

- CryCommon/CryExtension/ICryFactoryRegistry.h

**Additional Extensions**

Use the methods defined in ICryUnknown for additional functionality.

**Interface casting semantics**

Interface casting semantics have been implemented to provide syntactically convenient and type-safe casting of interfaces. The syntax was designed to conform with traditional C++ type casts and respects const rules.

```cpp
ICryFactory* pFactory = ...;
assert(pFactory);

ICryUnknownPtr pUnk = pFactory->CreateClassInstance();
IMyExtensionPtr pMyExtension = cryinterface_cast<IMyExtension>(pUnk);
if (pMyExtension)
{
    // it's safe to work with pMyExtension
}
```

Interface casting also works on raw interface pointers. However, please consider the guidelines described in the section Shared and Raw Interface Pointers (p. 170).

Declared in the following framework header file:

- CryCommon/CryExtension/ICryUnknown.h

**Querying interface identifiers**

Occasionally, it is necessary to know the ID of an interface, e.g. to pass it to ICryFactoryRegistry::IterateFactories(). This can be done as follows.
CryInterfaceID iid = cryiidof<IMyExtension>();

Declared in the following framework header file:

• CryCommon/CryExtension/ICryUnknown.h

Checking pointers

Use this extension to check whether pointers to different interfaces belong to the same class instance.

IMyExtensionAPtr pA = ...;
IMyExtensionBPtr pB = ...;
if (CryIsSameClassInstance(pA, pB))
{
 ...
}

This works on both shared and raw interface pointers.

Declared in the following framework header file:

• CryCommon/CryExtension/ICryUnknown.h

Querying composites

Extensions can be queried for composites as follows.

IMyExtensionPtr pMyExtension = ...;
ICryUnknownPtr pCompUnk = crycomposite_query(pMyExtension, "foo");
IFooPtr pComposite = cryinterface_cast<IFoo>(pCompUnk);
if (pComposite)
{
 // it's safe to work with pComposite, a composite of pMyExtension exposed as "foo"
 // implementing IFoo
}

A call to crycomposite_query() might return NULL if the specified composite has not yet been created. To gather more information, the query can be rewritten as follows.

IMyExtensionPtr pMyExtension = ...
bool exposed = false;
ICryUnknownPtr pCompUnk = crycomposite_query(pMyExtension, "foo", &exposed);
if (exposed)
{
 if (pCompUnk)
{
 // "foo" exposed and created

 IFooPtr pComposite = cryinterface_cast<IFoo>(pCompUnk);
 if (pComposite)
{
 // it's safe to work with pComposite, a composite of pMyExtension exposed as "foo"
 // implementing IFoo
}
As with interface casting composite, queries work on raw interface pointers. However, please consider the guidelines described in the section Shared and Raw Interface Pointers (p. 170).

Declared in the following framework header file:

• CryCommon/CryExtension/ICryUnknown.h

## Glue Code Macros

The following macros provide glue code to implement the base interfaces and services to support the framework in a thread-safe manner. You are strongly encouraged to use them when you implement an extension.

For examples of how these macros work together, see Using Glue Code (p. 183).

Declared in the following framework header files:

• CryCommon/CryExtension/Impl/ClassWeaver.h
• CryCommon/CryExtension/CryGUID.h

### CRYINTERFACE_DECLARE(iname, iidHigh, iidLow)

Declares an interface and associated ID. Protects the interfaces from accidentally being deleted on client side. That is, it allows destruction only by using `boost::shared_ptr<T>`. This macro is required once per interface declaration.

#### Parameters

iname

The (C++) name of the interface as declared.

iidHigh

The higher 64-bit part of the interface ID (GUID).

iidLow

The lower 64-bit part of the interface ID (GUID).

### CRYINTERFACE_BEGIN()

Start marker of the interface list inside the extension class implementation. Required once per extension class declaration.
CRYINTERFACE_ADD(iname)

Marker to add interfaces inside the extension class declaration. It has to be declared in between
CRYINTERFACE_BEGIN() and any of the CRYINTERFACE_END*() markers. Only declare the interfaces
that the class directly inherits. If deriving from an existing extension class or classes, the inherited
interfaces get added automatically. If an interface is declared multiple times, duplicates will be removed.
It is not necessary to add ICryUnknown.

Warning
Other interfaces that are not declared will not be castable by using
cryinterface_cast<T>().

Parameters

iname
The (C++) name of the interface to be added.

CRYINTERFACE_END()

End marker of the interface list inside the extension class declaration. Use this if not inheriting from any
already existing extension class. Required once per extension class declaration. Mutually exclusive with
any of the other CRYINTERFACE_END*() markers.

CRYINTERFACE_ENDWITHBASE(base)

End marker of the interface list inside the extension class declaration. Use this if inheriting from an
already existing extension class. Required once per extension class declaration. Mutually exclusive with
any of the other CRYINTERFACE_END*() markers.

Parameters

base
The (C++) name of the extension class from which derived.

CRYINTERFACE_ENDWITHBASE2(base0, base1)

End marker of the interface list inside the extension class declaration. Use this if inheriting from two
already existing extension classes. Required once per extension class declaration. Mutually exclusive with
any of the other CRYINTERFACE_END*() markers.

Parameters

base0
The (C++) name of the first extension class from which derived.

base1
The (C++) name of the second extension class from which derived.

CRYINTERFACE_ENDWITHBASE3(base0, base1, base2)

End marker of the interface list inside the extension class declaration. Use this if inheriting from three
already existing extension classes. Required once per extension class declaration. Mutually exclusive with
any of the other CRYINTERFACE_END*() markers.
Parameters

base0
- The (C++) name of the first extension class from which derived.

base1
- The (C++) name of the second extension class from which derived.

base2
- The (C++) name of the 3rd extension class from which derived.

CRYINTERFACE_SIMPLE(iname)
Convenience macro for the following code sequence (probably the most common extension case):

```cpp
CRYINTERFACE_BEGIN()
CRYINTERFACE_ADD(iname)
CRYINTERFACE_END()
```

Parameters

iname
- The (C++) name of the interface to be added.

CRYCOMPOSITE_BEGIN()
Start marker of the list of exposed composites.

CRYCOMPOSITE_ADD(member, membername)
Marker to add a member of the extension class to the list of exposed composites.

Parameters

member
- The (C++) name of the extension class member variable to be exposed. It has to be of type `boost::shared_ptr<T>`, where `T` inherits from `ICryUnknown`. This condition is enforced at compile time.

membername
- The name (as C-style string) of the composite by which the composite can later be queried at runtime.

CRYCOMPOSITE_END(implclassname)
End marker of the list of exposed composites. Use this if not inheriting from any extension class that also exposes composites. Mutually exclusive with any of the other CRYCOMPOSITE_END*() markers.

Parameters

implclassname
- The (C++) name of the extension class to be implemented.
CRYCOMPOSITE_ENDWITHBASE(implclassname, base)

End marker of the list of exposed composites. Use this if inheriting from one extension class that also exposes composites. Queries will first search in the current class and then look into the base class to find a composite that matches the requested name specified in crycomposite_query(). Mutually exclusive with any of the other CRYCOMPOSITE_END*() markers.

Parameters

implclassname

The (C++) name of the extension class to be implemented.

base

The (C++) name of the extension class derived from.

CRYCOMPOSITE_ENDWITHBASE2(implclassname, base0, base1)

End marker of the list of exposed composites. Use this if inheriting from two extension classes that also expose composites. Queries will first search in the current class and then look into the base classes to find a composite matching the requested name specified in crycomposite_query(). Mutually exclusive with any of the other CRYCOMPOSITE_END*() markers.

Parameters

implclassname

The (C++) name of the extension class to be implemented.

base0

The (C++) name of the first extension class from which derived.

base1

The (C++) name of the second extension class which derived.

CRYCOMPOSITE_ENDWITHBASE3(implclassname, base0, base1, base2)

End marker of the list of exposed composites. Use this if inheriting from three extension classes that also expose composites. Queries will first search in the current class and then look into the base classes to find a composite matching the requested name specified in crycomposite_query(). Mutually exclusive with any of the other CRYCOMPOSITE_END*() markers.

Parameters

implclassname

The (C++) name of the extension class to be implemented.

base0

The (C++) name of the first extension class from which derived.

base1

The (C++) name of the second extension class from which derived.
The (C++) name of the third extension class from which derived.

**CRYGENERATE_CLASS(implclassname, cname, cidHigh, cidLow)**

Generates code to support base interfaces and services for an extension class that can be instantiated an arbitrary number of times. Required once per extension class declaration. Mutually exclusive to CRYGENERATE_SINGLETONCLASS().

**Parameters**

- **implclassname**
  - The C++ class name of the extension.

- **cname**
  - The extension class name with which it is registered in the registry.

- **cidHigh**
  - The higher 64-bit part of the extension's class ID (GUID) with which it is registered in the registry.

- **cidLow**
  - The lower 64-bit part of the extension's class ID (GUID) with which it is registered in the registry.

**CRYGENERATE_SINGLETONCLASS(implclassname, cname, cidHigh, cidLow)**

Generates code to support base interfaces and services for an extension class that can be instantiated only once (singleton). Required once per extension class declaration. Mutually exclusive with CRYGENERATE_CLASS().

**Parameters**

- **implclassname**
  - The C++ class name of the extension.

- **cname**
  - The extension class name with which it is registered in the registry.

- **cidHigh**
  - The higher 64-bit part of the extension's class ID (GUID) with which it is registered in the registry.

- **cidLow**
  - The lower 64-bit part of the extension's class ID (GUID) with which it is registered in the registry.

**CRYREGISTER_CLASS(implclassname)**

Registers the extension class in the system. Required once per extension class at file scope.

**Parameters**

- **implclassname**
  - The C++ class name of the extension.
MAKE_CRYGUID(high, low)

Parameters

Constructs a CryGUID. Useful when searching the registry for extensions by class ID.

high

The higher 64-bit part of the GUID.

low

The lower 64-bit part of the GUID.

CryExtension Samples

Sample 1 - Implementing a Source Control Plugin by Using Extensions

```cpp
Thông điệp /// source control interface

struct ISourceControl : public ICryUnknown
{
    CRYINTERFACE_DECLARE(ISourceControl, 0x399d8fc1d94044cc, 0xa70d2b4e58921453)

    virtual void GetLatest(const char* filename) = 0;
    virtual void Submit() = 0;
};

typedef cryshared_ptr<ISourceControl> ISourceControlPtr;

 Sabha /// concrete implementations of source control interface

class CSourceControl_Perforce : public ISourceControl
{
    CRYINTERFACE_BEGIN()
    CRYINTERFACE_ADD(ISourceControl)
    CRYINTERFACE_END()

    CRYREGISTER_CLASS(CSourceControl_Perforce)

    CRYGENERATE_SINGLETONCLASS(CSourceControl_Perforce, "CSourceControl_Perforce",
    0x7305bff20ee543e3, 0x820792c56e74ecda)

    virtual void GetLatest(const char* filename) { ...);
    virtual void Submit() { ... });

    CRYREGISTER_CLASS(CSourceControl_Perforce)

    class CSourceControl_SourceSafe : public ISourceControl
    {
    CRYINTERFACE_BEGIN()
    CRYINTERFACE_ADD(ISourceControl)
    CRYINTERFACE_END()

    CRYREGISTER_CLASS(CSourceControl_SourceSafe)

    CRYGENERATE_SINGLETONCLASS(CSourceControl_SourceSafe, "CSourceControl_SourceSafe",
    0x1df62628db9d4bb2, 0x8164e418dd5b6691)
```
Using Extensions

Working with Specific Extension Classes

To work with a specific extension class, a client needs to know the extension's class name or class id and
the interface(s) that the class supports. With this information, the class factory can be queried from the
registry, an instance created and worked with as in the following example.

// IMyExtension.h
#include <CryExtension/ICryUnknown.h>

struct IMyExtension : public ICryUnknown
{
  ...
};

typedef boost::shared_ptr<IMyExtension> IMyExtensionPtr;

// in client code
#include <IMyExtension.h>
#include <CryExtension/CryCreateClassInstance.h>

IMyExtensionPtr pMyExtension;

#if 0
// create extension by class name
if (CryCreateClassInstance("MyExtension", pMyExtension))
#else
// create extension by class id, guaranteed to create instance of same kind
if (CryCreateClassInstance(MAKE_CRYGUID(0x68c7f0e0c36446fe, 0x82a3bc01b54dc7bf), pMyExtension))
#endif
// it's safe to work with pMyExtension
}

// verbose version of client code above
#include <IMyExtension.h>
#include <CryExtension/ICryFactory.h>
#include <CryExtension/ICryFactoryRegistry.h>

ICryFactoryRegistry* pReg = ...;

#if 0
// search extension by class name
ICryFactory* pFactory = pReg->GetFactory("MyExtension");
#else
// search extension by class id, guaranteed to yield same factory as in search by class name
ICryFactory* pFactory = pReg->GetFactory(MAKE_CRYGUID(0x68c7f0e0c36446fe, 0x82a3bc01b54dc7bf));
#endif

if (pFactory) // see comment below
{
  ICryUnknownPtr pUnk = pFactory->CreateClassInstance();
  IMyExtensionPtr pMyExtension = cryinterface_cast<IMyExtension>(pUnk);
  if (pMyExtension)
  {
    // it's safe to work with pMyExtension
  }
}

As an optimization, you can enhance the if check as follows.

if (pFactory && pFactory->ClassSupports(cryiidof<IMyExtension>()))
{
  ...
}

This version of the if statement will check interface support before the extension class is instantiated. This check prevents the unnecessary (and potentially expensive) construction and destruction of extensions that are incompatible with a given interface.

**Finding Extension Classes that Support a Specific Interface**

To determine how many extension classes in the registry support a given interface, and to list them, clients can submit queries similar to the following.

// IMyExtension.h
#include <CryExtension/ICryUnknown.h>

struct IMyExtension : public ICryUnknown
{
  ...
};

// in client code
#include <IMyExtension.h>
#include <CryExtension/ICryFactory.h>
#include <CryExtension/ICryFactoryRegistry.h>

ICryFactoryRegistry* pReg = ...;

size_t numFactories = 0;
Implementing Extensions Using the Framework

The following section explains in detail how to implement extensions in Lumberyard. It provides examples that use glue code and do not use glue code. The section also shows you how to utilize the framework in cases where ICryUnknown cannot be the base of the extension interface.

Recommended Layout for Including Framework Header Files

The public interface header that will be included by the client should look like the following.

```cpp
// IMyExtension.h
#include <CryExtension/ICryUnknown.h>
struct IMyExtension : public ICryUnknown
{
    ...
};
```

If you are using glue code, declare the implementation class of the extension in the header file as follows.

```cpp
// MyExtension.h
#include <IMyExtension.h>
#include <CryExtension/Impl/ClassWeaver.h>
class CMyExtension : public IMyExtension
{
    ...
};
```

Using Glue Code

The first example shows a possible implementation of the IMyExtension class in the previous examples.

```cpp
///////////////////////////// // public section
// IMyExtension.h
#include <CryExtension/ICryUnknown.h>
struct IMyExtension : public ICryUnknown
{
    CRYINTERFACE_DECLARE(IMyExtension, 0x4fb87a5f83f74323, 0xa7e42ca947c549d8)
    virtual void CallMe() = 0;
};
typedef boost::shared_ptr<IMyExtension> IMyExtensionPtr;
```
The following example shows how the extension class `MyExtension` can be customized and expanded to implement two more interfaces, `IFoo` and `IBar`.
Without Using Glue Code

If for any reason using the glue code is neither desired nor applicable, extensions can be implemented as follows. It is recommended to implement ICryUnknown and ICryFactory such that their runtime cost is equal to the one provided by the glue code. For more information, see ICryUnknown (p. 171) and ICryFactory (p. 171).
// INoMacros.h
#include <CryExtension/ICryUnknown.h>

struct INoMacros : public ICryUnknown
{
  // befriend cryiidof and boost::checked_delete
  template <class T> friend const CryInterfaceID& InterfaceCastSemantics::cryiidof();
  template <class T> friend void boost::checked_delete(T* x);
  protected:
    virtual ~INoMacros() {}  

private:
  // It's very important that this static function is implemented for each interface!
  // Otherwise the consistency of cryinterface_cast<T>() is compromised because
  // cryiidof<T>() = cryiidof<baseof<T>>() {baseof<T> = ICryUnknown in most cases}
  static const CryInterfaceID IID()
  {
    static const CryInterfaceID iid = {0xd0fda1427dee4cceull, 0x88ff91b6b7be2a1full};
    return iid;
  }

public:
  virtual void TellMeWhyIDontLikeMacros() = 0;
};

typedef boost::shared_ptr<INoMacros> INoMacrosPtr;

// private section not visible to client

// NoMacros.cpp

// This is just an exemplary implementation!
// For brevity the whole implementation is packed into this cpp file.
#include <INoMacros.h>
#include <CryExtension/ICryFactory.h>
#include <CryExtension/Impl/RegFactoryNode.h>

// implement factory first
class CNoMacrosFactory : public ICryFactory
{
  // ICryFactory
  public:
    virtual const char* GetClassName() const
    {
      return "NoMacros";
    }
    virtual const CryClassID& GetClassID() const
    {
      static const CryClassID cid = {0xa4550317690145c1ull, 0xa7eb5d85403dfad4ull};
      return cid;
    }
    virtual ICryUnknownPtr CreateClassInstance() const;
  public:

static CNoMacrosFactory& Access()
{
    return s_factory;
}

private:
    CNoMacrosFactory() {}
    ~CNoMacrosFactory() {}

private:
    static CNoMacrosFactory s_factory;
};
CNoMacrosFactory CNoMacrosFactory::s_factory;

// implement extension class
class CNoMacros : public INoMacros
{
    // ICryUnknown
public:
    virtual ICryFactory* GetFactory() const
    {
        return &CNoMacrosFactory::Access();
    }

    // befriend boost::checked_delete
    // only needed to be able to create initial shared_ptr<CNoMacros>
    // so we don’t lose type info for debugging (i.e. inspecting shared_ptr)
    template <class T> friend void boost::checked_delete(T* x);

protected:
    virtual void* QueryInterface(const CryInterfaceID& iid) const
    {
        if (iid == cryiidof<ICryUnknown>())
            return (void*) (ICryUnknown*) this;
        else if (iid == cryiidof<INoMacros>())
            return (void*) (INoMacros*) this;
        else
            return 0;
    }

    virtual void* QueryComposite(const char* name) const
    {
        return 0;
    }

    // INoMacros
public:
    virtual void TellMeWhyIDontLikeMacros()
    {
        printf("Woohoo, no macros...\n");
    }
    CNoMacros() {}

protected:
    virtual ~CNoMacros() {};
};

// implement factory’s CreateClassInstance method now that extension class is fully visible
to compiler
ICryUnknownPtr CNoMacrosFactory::CreateClassInstance() const
{
    boost::shared_ptr<CNoMacros> p(new CDontLikeMacros);
    return ICryUnknownPtr(*static_cast<boost::shared_ptr<ICryUnknown>*>(static_cast<void*>(&p)));
Exposing Composites

The following example shows how to expose (inherited) composites. For brevity, the sample is not separated into files.

```cpp
// Exposing Composites

struct ITestExt1 : public ICryUnknown
{
    CRYINTERFACE_DECLARE(ITestExt1, 0x9d9e0dcfa5764cb0, 0xa73701595f75bd32)
    virtual void Call1() = 0;
};

typedef boost::shared_ptr<ITestExt1> ITestExt1Ptr;

class CTestExt1 : public ITestExt1
{
    CRYINTERFACE.Begin()
        CRYINTERFACE.Add(ITestExt1)
    CRYINTERFACE.End()

    CRYGENERATE_CLASS(CTestExt1, "TestExt1", 0x43b04e7cc1be45ca, 0x9df6ccb1c0dc1ad8)

    public:
        virtual void Call1();
};

CRYREGISTER_CLASS(CTestExt1)

CTestExt1::CTestExt1()
{
}

CTestExt1::~CTestExt1()
{
}

void CTestExt1::Call1()
{
}

class CComposed : public ICryUnknown
{
    CRYINTERFACE.Begin()
    CRYINTERFACE.End()

    CRYCOMPOSITE_BEGIN()
        CRYCOMPOSITE.Add(m_pTestExt1, "Ext1")
    CRYCOMPOSITE.End(CComposed)

    CRYGENERATE_CLASS(CComposed, "Composed", 0x0439d74b8dcd4b7f, 0x9287dcdf7e26a3a5)

    private:
        ITestExt1Ptr m_pTestExt1;
};

CRYREGISTER_CLASS(CComposed)
```
CComposed::CComposed()
: m_pTestExt1()
{
    CryCreateClassInstance("TestExt1", m_pTestExt1);
}

CComposed::~CComposed()
{
}

//////////////////////////////////////////////////////////////////////////
struct ITestExt2 : public ICryUnknown
{
    CRYINTERFACE_DECLARE(ITestExt2, 0x8eb7a4b399874b9c, 0xb96bd6da7a8c72f9)
    virtual void Call2() = 0;
};
DECLARE_BOOST_POINTERS(ITestExt2);

class CTestExt2 : public ITestExt2
{
    CRYINTERFACE_BEGIN()
    CRYINTERFACE_ADD(ITestExt2)
    CRYINTERFACE_END()

    CRYGENERATE_CLASS(CTestExt2, "TestExt2", 0x25b3ebf8f1754b9a, 0xb5494e3da7cdd80f)

    public:
    virtual void Call2();
};
CRYREGISTER_CLASS(CTestExt2)

CTestExt2::CTestExt2()
{
}

CTestExt2::~CTestExt2()
{
}

void CTestExt2::Call2()
{
}

//////////////////////////////////////////////////////////////////////////
class CMultiComposed : public CComposed
{
    CRYCOMPOSITE_BEGIN()
    CRYCOMPOSITE_ADD(m_pTestExt2, "Ext2")
    CRYCOMPOSITE_ENDWITHBASE(CMultiComposed, CComposed)

    CRYGENERATE_CLASS(CMultiComposed, "MultiComposed", 0x0419d74b8dcd4b7e, 0x9287dcdf7e26a3a6)

    private:
    ITestExt2Ptr m_pTestExt2;
};
CRYREGISTER_CLASS(CMultiComposed)

CMultiComposed::CMultiComposed()
: m_pTestExt2()
{
    CryCreateClassInstance("TestExt2", m_pTestExt2);
Implementing Extensions Using the Framework

CMultiComposed::~CMultiComposed()
{
}
...

//////////////////////////////////////////////////////////////////////////
// let's use it
ICryUnknownPtr p;
if (CryCreateClassInstance("MultiComposed", p))
{
  ITestExt1Ptr p1 = cryinterface_cast<ITestExt1>(crycomposite_query(p, "Ext1"));
  if (p1)
    p1->Call1(); // calls CTestExt1::Call1()
  ITestExt2Ptr p2 = cryinterface_cast<ITestExt2>(crycomposite_query(p, "Ext2"));
  if (p2)
    p2->Call2(); // calls CTestExt2::Call2()
}

If ICryUnknown Cannot Be the Base of the Extension Class

There are cases where making ICryUnknown the base of your extension class is not possible. Some examples are legacy code bases that cannot be modified, third party code for which you do not have full source code access, or code whose modification is not practical. However, these code bases can provide useful functionality (for example, for video playback or flash playback) if you expose them as engine extensions. The following sample illustrates how an additional level of indirection can expose a third party API.

///////////////////////////////////////////
// public section
#include <IExposeThirdPartyAPI.h>
#include <IThirdPartyAPI.h>
struct IExposeThirdPartyAPI : public ICryUnknown
{
  CRYINTERFACE_DECLARE(IXposeThirdPartyAPI, 0x804250bbaacf4a5f, 0x90ef0327bb7a0a7f)
    virtual IThirdPartyAPI* Create() = 0;
};
typedef boost::shared_ptr<IXposeThirdPartyAPI> IExposeThirdPartyAPIPtr;

///////////////////////////////////////////
// private section not visible to client
#include <IXposeThirdPartyAPI.h>
#include <CryExtension/Impl/ClassWeaver.h>
class CExposeThirdPartyAPI : public IExposeThirdPartyAPI
{
  CRYINTERFACE_BEGIN()
    CRYINTERFACE_ADD(IXposeThirdPartyAPI)
  CRYINTERFACE_END()
  CRYGENERATE_CLASS(CExposeThirdPartyAPI, "ExposeThirdPartyAPI", 0xa93b970b2c434a21, 0x86acfe94d8daes547)
public:
    virtual IThirdPartyAPI* Create();
};

// ExposeThirdPartyAPI.cpp
#include "ExposeThirdPartyAPI.h"
#include "ThirdPartyAPI.h"
CRYREGISTER_CLASS(CExposeThirdPartyAPI)
CExposeThirdPartyAPI::CExposeThirdPartyAPI()
{
}
CExposeThirdPartyAPI::~CExposeThirdPartyAPI()
{
}
IThirdPartyAPI* CExposeThirdPartyAPI::Create()
{
    return new CThirdPartyAPI; // CThirdPartyAPI implements IThirdPartyAPI
}

Custom Inclusion and Exclusion of Extensions

To enable easy inclusion and exclusion of extensions, Lumberyard provides a global "extension
definition" header much like CryCommon/ProjectDefines.h that is automatically included in all
modules by means of the platform.h file. To wrap your extension implementation code, you include
a #define statement in the extension definition header. To exclude unused extension code from your
build, you can also comment out extensions that you are not interested in. Interface headers are not
affected by the #if defined statements, so the client code compiles as is with or without them.

////////////////////////////////////////////////////////////////////
// public section

// IMyExtension.h
#include <CryExtension/ICryUnknown.h>

struct IMyExtension : public ICryUnknown
{
    ...
};
typedef boost::shared_ptr<IMyExtension> IMyExtensionPtr;

// ExtensionDefines.h
...
#define INCLUDE_MYEXTENSION
...
////////////////////////////////////////////////////////////////////
// private section not visible to client

// MyExtension.h
#if defined(INCLUDE_MYEXTENSION)

#include <IMyExtension.h>
#include <CryExtension/Impl/ClassWeaver.h>

class CMyExtension : public IMyExtension
{
    ...

191
Because extensions can be removed from a build, clients must write their code in a way that does not assume the availability of an extension. For more information, see Using Extensions (p. 181).

### CryString

**Note**
The use of CryString has been deprecated in Lumberyard. Instead, use the AZStd::string class. AZStd::string has the same interface as std::string in C++ but, like most datastructures in AZStd, can be serialized AZ::SerializeContext. For some useful string functions, see the file \dev\Code\Framework\AzFramework\AzFramework\StringFunc\StringFunc.h. AZStd::String also has a format(...) function which you can use to create a formatted string, as in the following example:

```cpp
AZStd::string myFormattedString = AZStd::string::format("My formatted string with number \%d", 5);
```

The legacy custom reference-counted string class CryString (declared in CryString.h) is a replacement for STL std::string. CryString should always be preferred over std::string. For convenience, string is used as a typedef for CryString.

### How to Use Strings as Key Values for STL Containers

The following code shows good (efficient) and bad usage:

```cpp
const char *szKey= "Test";

map< string, int >::const_iterator iter = m_values.find( CONST_TEMP_STRING( szKey ) ); // Good

map< string, int >::const_iterator iter = m_values.find( szKey ); // Bad
```

By using the suggested method, you avoid the allocation, deallocation, and copying of a temporary string object, which is a common problem for most string classes. By using the macro CONST_TEMP_STRING, the string class uses the pointer directly without having to free data afterwards.

### Further Usage Tips

- Do not use std::string or std::wstring. Instead, use only string and wstring, and never include the standard string header <string>.
• Use the `c_str()` method to access the contents of the string.
• Because strings are reference-counted, never modify memory returned by the `c_str()` method. Doing so could affect the wrong string instance.
• Do not pass strings via abstract interfaces; all interfaces should use `const char*` in interface methods.
• `CryString` has a combined interface of `std::string` and the MFC `CString`, so you can use both interface types for string operations.
• Avoid doing many string operations at runtime as they often cause memory reallocations.
• For fixed size strings (e.g. 256 chars), use `CryFixedStringT`, which should be preferred over static char arrays.

### ICrySizer

The `ICrySizer` interface can be implemented to record detailed information about the memory usage of a class.

**Note**
This information is also available in the Editor under **Engine Memory info**.

### How to use the ICrySizer interface

The following example shows how to use the `ICrySizer` interface.

```cpp
void GetMemoryUsage( ICrySizer *pSizer )
{
    SIZER_COMPONENT_NAME( pSizer, "Renderer (Aux Geometries)" );
    pSizer->Add(*this);
}
pSizer->AddObject(<element_prow>,<element_count>);
pSizer->AddObject(<container>);
m_SubObject.GetMemoryUsage(pSizer);
```

### Legacy Serialization Library

**Note**
The legacy serialization library has been replaced in Lumberyard by the `AZ::SerializeContext`.

The `CryCommon` serialization library has the following features:

• Separation of user serialization code from the actual storage format. This makes it possible to switch between XML, JSON, and binary formats without changing user code.
• Re-usage of the same serialization code for editing in the PropertyTree. You can write the serialization code once and use it to expose your structure in the editor as a parameters tree.
• Enables you to write serialization code in non-intrusive way (as global overloaded functions) without modifying serialized types.
• Makes it easy to change formats. For example, you can add, remove, or rename fields and still be able to load existing data.
Tutorial

The example starts with a data layout that uses standard types, enumerations, and containers. The example adds the Serialize method to structures with fixed signatures.

Defining data

```
#include "Serialization/IArchive.h"
#include "Serialization/STL.h"

enum AttachmentType
{
    ATTACHMENT_SKIN,
    ATTACHMENT_BONE
};
struct Attachment
{
    string name;
    AttachmentType type;
    string model;
    void Serialize(Serialization::IArchive& ar)
    {
        ar(name, "name", "Name");
        ar(type, "type", "Type");
        ar(model, "model", "Model");
    }
};
struct Actor
{
    string character;
    float speed;
    bool alive;
    std::vector<Attachment> attachments;
    Actor()
    : speed(1.0f)
    , alive(true)
    {
    }
    void Serialize(Serialization::IArchive& ar)
    {
        ar(character, "character", "Character");
        ar(speed, "speed", "Speed");
        ar(alive, "alive", "Alive");
        ar(attachments, "attachments", "Attachment");
    }
};

// Implementation file:
#include "Serialization/Enum.h"
SERIALIZATION_ENUM_BEGIN(AttachmentType, "Attachment Type")
SERIALIZATION_ENUM(ATTACHMENT_BONE, "bone", "Bone")
SERIALIZATION_ENUM(ATTACHMENT_SKIN, "skin", "Skin")
SERIALIZATION_ENUM_END()
```

Why are two names needed?

The ar() call takes two string arguments: one is called name, and the second label. The name argument is used to store parameters persistently; for example, for JSON and XML. The label parameter is used for the PropertyTree. The label parameter is typically longer, more descriptive, contains white space, and may be easily changed without breaking compatibility with existing data. In
contrast, name is a C-style identifier. It is also convenient to have name match the variable name so that developers can easily find the variable by looking at the data file.

Omitting the label parameter (the equivalent of passing nullptr) will hide the parameter in the PropertyTree, but it will be still serialized and can be copied together with its parent by using copy-paste.

**Note**
The SERIALIZATION_ENUM macros should reside in the .cpp implementation file because they contain symbol definitions.

### Serializing into or from a file

Now that the data has been defined, it is ready for serialization. To implement the serialization, you can use Serialization::SaveJsonFile, as in the following example.

```cpp
#include <Serialization/IArchiveHost.h>

Actor actor;
Serialization::SaveJsonFile("filename.json", actor);
```

This will output content in the following format:

```json
{
  "character": "nanosuit.cdf",
  "speed": 2.5,
  "alive": true,
  "attachments": [
    { "name": "attachment 1", "type": "bone", "model": "model1.cgf" },
    { "name": "attachment 2", "type": "skin", "model": "model2.cgf" }
  ]
}
```

The code for reading data is similar to that for serialization, except that it uses Serialization::LoadJsonFile.

```cpp
#include <Serialization/IArchiveHost.h>

Actor actor;
Serialization::LoadJsonFile(actor, "filename.json");
```

The save and load functions used are wrappers around the IArchiveHost interface, an instance of which is located in gEnv->pSystem->GetArchiveHost(). However, if you have direct access to the archive code (for example, in CrySystem or EditorCommon), you can use the archive classes directly, as in the following example.

```cpp
#include <Serialization/JSONOArchive.h>
#include <Serialization/JSONIArchive.h>

Serialization::JSONOArchive oa;
Actor actor;
oa(actor);
oa.save("filename.json");

// to get access to the data without saving:
const char* jsonString = oa.c_str();

// and to load
Serialization::JSONIArchive ia;
if (ia.load("filename.json"))
```
Editing in the PropertyTree

If you have the Serialize method implemented for your types, it is easy to get it exposed to the QPropertyTree, as the following example shows.

```cpp
#include <QPropertyTree/QPropertyTree.h>
QPropertyTree* tree = new QPropertyTree(parent);
static Actor actor;
tree->attach(Serialization::SStruct(actor));
```

You can select enumeration values from the list and add or remove vector elements by using the [ 2 ] button or the context menu.

In the moment of attachment, the Serialize method will be called to extract properties from your object. As soon as the user changes a property in the UI, the Serialize method is called to write properties back to the object.

**Note**
It is important to remember that QPropertyTree holds a reference to an attached object. If the object's lifetime is shorter than the tree, an explicit call to QPropertyTree::detach() should be performed.

Use Cases

Non-intrusive serialization

Normally when struct or a class instance is passed to the archive, the Serialize method of the instance is called. However, it is possible to override this behavior by declaring the following global function:

```cpp
bool Serialize(Serialization::IArchive&, Type& value, const char* name, const char* label);
```

The return value here has the same behavior as IArchive::operator(). For input archives, the function returns false when a field is missing or wasn't read. For output archives, it always returns true.
Note
The return value does not propagate up. If one of the nested fields is missing, the top level
block will still return true.

The global function approach is useful when you want to:

- Add serialization in non-intrusive way
- Transform data during serialization
- Add support for unsupported types like plain pointers

The following example adds support for `std::pair<>` type to the `Serialize` function:

```cpp
template<class T1, class T2>
struct pair_serializable : std::pair<T1, T2>
{
    void Serialize(Serialization::IArchive& ar)
    {
        ar(first, "first", "First");
        ar(second, "second", "Second");
    }
};

template<class T1, class T2>
bool Serialize(Serialization::IArchive& ar, std::pair<T1, T2>& value, const char* name,
        const char* label)
{
    return ar(static_cast<pair_serializable<T1, T2>&>(value), name, label);
}
```

The benefit of using inheritance is that you can get access to protected fields. In cases when access
policy is not important and inheritance is undesirable, you can replace the previous code with following
pattern.

```cpp
template<class T1, class T2>
struct pair_serializable
{
    std::pair<T1, T2>& instance;

    pair_serializable(std::pair<T1, T2>&& instance) : instance(instance) {};

    void Serialize(Serialization::IArchive& ar)
    {
        ar(instance.first, "first", "First");
        ar(instance.second, "second", "Second");
    }
};

template<class T1, class T2>
bool Serialize(Serialization::IArchive& ar, std::pair<T1, T2>& value, const char* name,
        const char* label)
{
    pair_serializable<T1, T2> serializer(value);
    return ar(serializer, name, label);
}
```

Registering Enum inside a Class

Normally, `SERIALIZATION_ENUM_BEGIN()` will not compile if you specify enumeration within a class
(a "nested enum"). To overcome this shortcoming, use `SERIALIZATION_ENUM_BEGIN_NESTED`, as in the
following example.
Polymorphic Types

The Serialization library supports the loading and saving of polymorphic types. This is implemented through serialization of a smart pointer to the base type.

For example, if you have following hierarchy:

IBase

- ImplementationA
- ImplementationB

You would need to register derived types with a macro, as in the following example.

```cpp
SERIALIZATION_CLASS_NAME(IBase, ImplementationA, "impl_a", "Implementation A");
SERIALIZATION_CLASS_NAME(IBase, ImplementationA, "impl_b", "Implementation B");
```

Now you can serialize a pointer to the base type:

```cpp
#include <Serialization/SmartPtr.h>
_smart_ptr<IInterface> pointer;
ar(pointer, "pointer", "Pointer");
```

The first string is used to name the type for persistent storage, and the second string is a human-readable name for display in the PropertyTree.

Customizing presentation in the PropertyTree

There are two aspects that can be customized within the PropertyTree:

1. The layout of the property fields. These are controlled by control sequences in the label (the third argument in `IArchive::operator()`).
2. Decorators. These are defined in the same way that specific properties are edited or represented.

Control characters

Control sequences are added as a prefix to the third argument for `IArchive::operator()`. These characters control the layout of the property field in the PropertyTree.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>Read-only</td>
<td>Prevents the user from changing the value of the property. The effect is non-recursive.</td>
</tr>
<tr>
<td>^</td>
<td>Inline</td>
<td>Places the property on the same line as the name of the structure root. Can be used to put fields in one line in a horizontal layout, rather than in the default vertical list.</td>
</tr>
</tbody>
</table>
### Use Cases

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>^^</td>
<td>Inline in front of a name</td>
<td>Places the property name before the name of the parent structure. Useful to add check boxes before a name.</td>
</tr>
<tr>
<td>&lt;</td>
<td>Expand value field</td>
<td>Expand the value part of the property to occupy all available space.</td>
</tr>
<tr>
<td>&gt;</td>
<td>Contract value field</td>
<td>Reduces the width of the value field to the minimum. Useful to restrict the width of inline fields.</td>
</tr>
<tr>
<td>&gt;\N&gt;</td>
<td>Limit field width to \N pixels</td>
<td>Useful for finer control over the UI. Not recommended for use outside of the editor.</td>
</tr>
<tr>
<td>+</td>
<td>Expand row by default.</td>
<td>Can be used to control which structures or containers are expanded by default. Use this only when you need per-item control. Otherwise, QPropertyTree::setExpandLevels is a better option.</td>
</tr>
<tr>
<td>[S]</td>
<td>Apply \S control characters to children.</td>
<td>Applies control characters to child properties. Especially useful with containers.</td>
</tr>
</tbody>
</table>

#### Combining control characters

Multiple control characters can be put together to combine their effects, as in the following example.

```cpp
ar(name, "name", "^!<Name"); // inline, read-only, expanded value field
```

#### Decorators

There are two kinds of decorators:

1. Wrappers that implement a custom serialization function that performs a transformation on the original value. For example, Serialization/Math.h contains `Serialization::RadiansAsDeg(float&)` that allows to store and edit angles in radians.
2. Wrappers that do no transformation but whose type is used to select a custom property implementation in the PropertyTree. Resource Selectors are examples of this kind of wrapper.

<table>
<thead>
<tr>
<th>Decorator</th>
<th>Purpose</th>
<th>Defined for types</th>
<th>Context needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnimationPath</td>
<td>Selection UI for full animation path.</td>
<td>Any string-like type, like: std::string, string (CryStringT), SCRCRef CCryName</td>
<td></td>
</tr>
<tr>
<td>CharacterPath</td>
<td>UI: browse for character path (cdf)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decorator</td>
<td>Purpose</td>
<td>Defined for types</td>
<td>Context needed</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>------------------------------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>CharacterRigPath</td>
<td>UI: browse for .rig files.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SkeletonPath</td>
<td>UI: browse for .chr or .skel files.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JointName</td>
<td>UI: list of character joints</td>
<td>ICharacterInstance*</td>
<td></td>
</tr>
<tr>
<td>AttachmentName</td>
<td>UI: list of character attachments</td>
<td>ICharacterInstance*</td>
<td></td>
</tr>
<tr>
<td>SoundName</td>
<td>UI: list of sounds</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ParticleName</td>
<td>UI: particle effect selection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Serialization/Decorators/Math.h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RadiansAsDeg</td>
<td>Edit or store radians as degrees</td>
<td>float, Vec3</td>
<td></td>
</tr>
<tr>
<td>Serialization/Decorators/Range.h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Range</td>
<td>Sets soft or hard limits for numeric values and provides a slider UI.</td>
<td>Numeric types</td>
<td></td>
</tr>
<tr>
<td>Serialization/Callback.h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Callback</td>
<td>Provides per-property callback function. See Adding callbacks to the PropertyTree (p. 201).</td>
<td>All types apart from compound ones (structs and containers)</td>
<td></td>
</tr>
</tbody>
</table>

**Decorator example**

The following example uses the Range and CharacterPath decorators.

```cpp
float scalar;
ar(Serialization::Range(scalar), 0.0f, 1.0f); // provides slider-UI
string filename;
ar(Serialization::CharacterPath(filename), "character", "Character"); // provides UI for file selection with character filter
```

**Serialization context**

The signature of the Serialize method is fixed. This can prevent the passing of additional arguments into nested Serialize methods. To resolve this issue, you can use a serialization context to pass a pointer of a specific type to nested Serialize calls, as in the following example.
void Scene::Serialize(Serialization::IArchive& ar)
{
    Serialization::SContext sceneContext(ar, this);
    ar(rootNode, "rootNode")
}

void Node::Serialize(Serialization::IArchive& ar)
{
    if (Scene* scene = ar.FindContext<Scene>())
    {
        // use scene
    }
}

Contexts are organized into linked lists. Nodes are stored on the stack within the SContext instance.

You can have multiple contexts. If you provide multiple instances of the same type, the innermost context will be retrieved.

You may also use contexts with the PropertyTree without modifying existing serialization code. The easiest way to do this is to use CContextList (QPropertyTree/ContextList.h), as in the following example.

    // CContextList m_contextList;
    tree = new QPropertyTree();
    m_contextList.Update<Scene>(m_scenePointer);
    tree->setArchiveContext(m_contextList.Tail());
    tree->attach(Serialization::SStruct(node));

Serializing opaque data blocks

It is possible to treat a block of data in the archive in an opaque way. This capability enables the Editor to work with data formats it has no knowledge of.

These data blocks can be stored within Serialization::SBlackBox. SBlackBox can be serialized or deserialized as any other value. However, when you deserialize SBlackBox from a particular kind of archive, you must serialize by using a corresponding archive. For example, if you obtained your SBlackBox from JSONIArchive, you must save it by using JSONOArchive.

Adding callbacks to the PropertyTree

When you change a single property within the property tree, the whole attached object gets deserialized. This means that all properties are updated even if only one was changed. This approach may seem wasteful, but has the following advantages:

- It removes the need to track the lifetime of nested properties, and the requirement that nested types be referenced from outside in safe manner.
- The content of the property tree is not static data, but rather the result of the function invocation. This allows the content to be completely dynamic. Because you do not have to track property lifetimes, you can serialize and de-serialize variables constructed on the stack.
- The removal of the tracking requirement results in a smaller amount of code.

Nevertheless, there are situations when it is desirable to know exactly which property changes. You can achieve this in two ways: 1) by using the Serialize method, or 2) by using Serialization::Callback.

1. Using the Serialize method, compare the new value with the previous value, as in the following example.
void Type::Serialize(IArchive& ar)
{
    float oldValue = value;
    ar(value, "value", "Value");
    if (ar.IsInput() && oldValue != value)
    {
        // handle change
    }
}

2. The second option is to use the Serialization::Callback decorator to add a callback function for one or more properties, as the following example illustrates.

```cpp
#include <Serialization/Callback.h>
using Serialization::Callback;

ar(Callback(value,
    [](float newValue) { /* handle change */ },
    "value", "Value");
```

**Note**

Callback works only with the PropertyTree, and should be used only in Editor code.

Callback can also be used together with other decorators, but in rather clumsy way, as the following example shows.

```cpp
ar(Callback(value,
    [](float newValue) { /* handle change */ },
    [](float& v) { return Range(v, 0.0f, 1.0f); },
    "value", "Value");
```

Of the two approaches, the callback approach is more flexible, but it requires you to carefully track the lifetime of the objects that are used by the callback lambda or function.

**PropertyTree in MFC window**

If your code base still uses MFC, you can use the PropertyTree with it by using a wrapper that makes this possible, as the following example shows.

```cpp
#include <IPropertyTree.h> // located in Editor/Include

int CMyWindow::OnCreate(LPCREATESTRUCT pCreateStruct)
{
    ...
    CRect clientRect;
    GetClientRect(clientRect);
    IPropertyTree* pPropertyTree = CreatePropertyTree(this, clientRect);
    ...
}
```

The IPropertyTree interface exposes the methods of QPropertyTree like Attach, Detach and SetExpandLevels.

**Documentation and validation**

QPropertyTree provides a way to add short documentation in the form of tool tips and basic validation.
The `Doc` method allows you to add tool tips to `QPropertyTree`, as in the following examples.

```cpp
void IArchive::Doc(const char*)

void SProjectileParameter::Serialize(IArchive& ar) {
    ar.Doc("Defines projectile physics.");
    ar(m_velocity, "velocity", "Velocity");
    ar.Doc("Defines initial velocity of the projectile.");
}
```

The `Doc` method adds a tool tip to last serialized element. When used at the beginning of the function, it adds the tool tip to the whole block.

The `Warning` and `Error` calls allow you to display warnings and error messages associated with specific property within the property tree, as in the following examples.

```cpp
template<class T> void IArchive::Warning(T& instance, const char* format, ...)
template<class T> void IArchive::Error(T& instance, const char* format, ...)
```

```cpp
void BlendSpace::Serialize(IArchive& ar) {
    ar(m_dimensions, "dimensions, "Dimensions");
    if (m_dimensions.empty())
        ar.Error(m_dimensions, "At least one dimension is required for BlendSpace");
}
```

The error message appears as follows.

![Error message example]

Warning messages look like this:

![Warning message example]

**Drop-down menu with a dynamic list**

If you want to specify an enumeration value, you can use the `enum` registration macro as described in the Defining data (p. 194) section.

There are two ways to define a drop-down menu: 1) transform your data into `Serialization::StringListValue`, or 2) implement a custom `PropertyRow` in the UI.

A short example of the first approach follows. The example uses a custom reference.

```cpp
// a little decorator that would annotate string as a special reference
struct MyReference {
    string& str;
```
MyReference(string& str) : str(str) {}
};

inline bool Serialize(Serialization::IArchive& ar, MyReference& wrapper, const char* name, const char* label)
{
 if (ar.IsEdit())
 { 
 Serialization::StringList items;
 items.push_back(""pageTitle");
 items.push_back("Item 1");
 items.push_back("Item 2");
 items.push_back("Item 3");
 Serialization::StringListValue dropDown(items, wrapper.str.c_str());
 if (!ar(dropDown, name, label))
 return false;
 if (ar.IsInput())
  wrapper.str = dropDown.c_str();
 return true;
 }
 else
 { 
  // when loading from disk we are interested only in the string
  return ar(wrapper.str, name, label);
 }
}

Now you can construct MyReference on the stack within the Serialize method to serialize a string as a dropdown item, as in the following example.

```c++
struct SType
{
  string m_reference;
  void SType::Serialize(Serialization::IArchive& ar)
  {
    ar(MyReference(m_reference), "reference", "Reference");
  }
};
```

The second way to define a drop-down menu requires that you implement a custom PropertyRow in the UI. This takes more effort, but makes it possible to create the list of possible items entirely within editor code.
CryNetwork Backward Compatibility

The deprecated legacy networking system called "CryNetwork" is no longer supported in Lumberyard and has been replaced by GridMate. To facilitate the transition of projects from CryNetwork to GridMate, Lumberyard provides a backward compatibility layer. The compatibility layer is encapsulated inside the CryNetwork library and exposed through the INetwork interface.

**Note**
Because the CryNetwork backward compatibility API layer uses CPU and bandwidth inefficiently, it is not advised for creating multiplayer games.

Systems that were added to provide backward compatibility for GridMate to the networked systems in CryEngine include remote method invocations, network serialization, and aspects.

**Topics**
- RMI Functions (p. 205)
- Network Serialization and Aspects (p. 207)

RMI Functions

To send remote method invocations (RMIs), use the InvokeRMI function, which has the following syntax:

```cpp
void InvokeRMI( IRMIRep& <rep>, ParamsType&& <params>, uint32 <where>, ChannelId <channel> = kInvalidChannelId );
```

**Parameters**

- `<rep>`
  
  Represents the remote function to be called (the RMI ID).

- `<params>`
  
  Specifies the parameters to pass into the remote function.

- `<where>`
  
  Specifies a flag that determines the category of clients to which the RMI will be sent. For information, see the RMI Function Flags (p. 206) section later in this document.

- `<channel>`
  
  Specifies specific clients to which the RMI will be sent, or specific clients to exclude. For information, see the RMI Function Flags (p. 206) section later in this document.

Ordering RMI Functions

The IGameObject.h file includes macros for declaring RMI classes (for example, those beginning with DECLARE_SERVER_RMI_<...>). The different declaration types are as follows:

- **PREATTACH** – The RMI is attached at the top of the data update for the object. You can use this declaration type to prepare the remote entity for new incoming data.
- **POSTATTACH** – The RMI is attached at the bottom of the data update, so it is called after the data is serialized. You can use this declaration type to complete an action with the new data.
• **NOATTACH** – The RMI is not attached to a data update, so the RMI cannot rely on the data. You can use this declaration type for calls that don't rely on data.

## Ordering Rules

The order for RMIs is only applicable within an object and attachment type set.

For example, in the following ordered list, PLAYER RMI 1, 2, and 3 will arrive in that order; however, ITEM RMI 1 might arrive before or after the following PLAYER RMIs:

- PLAYER RMI 1
- PLAYER RMI 2
- ITEM RMI 1
- PLAYER RMI 3

Using declaration types adds a layer of complication to the order of incoming data:

- **PREATTACH** – Messages are ordered within themselves.
- **POSTATTACH** – Messages are ordered within themselves.
- **NOATTACH** – Messages are ordered within themselves; however, **NOATTACH** can only fall on either side of the following diagram and never in between:

![Diagram](preattach_entityaspectdata_postattach)

### RMI Function Flags

To specify the clients that will receive an RMI, replace the `<where>` parameter in the `InvokeRMI` function with one of the following flags.

#### Server RMIs

- **eRMI_ToClientChannel**
  
  Sends an RMI from the server to a specific client. Specify the destination channel in the `<channel>` parameter.

- **eRMI_ToOwningClient**
  
  Sends an RMI from the server to the client that owns the actor.

- **eRMI_ToOtherClients**
  
  Sends an RMI from the server to all clients except the client specified. Specify the client to ignore in the `<channel>` parameter.

- **eRMI_ToRemoteClients**
  
  Sends an RMI from the server to all remote clients. Ignores the local client.

- **eRMI_ToOtherRemoteClients**
  
  Sends an RMI from the server to all remote clients except the remote client specified. Ignores the local client. The remote client to ignore is specified in the `<channel>` parameter.
eRMI_ToAllClients

Sends an RMI from the server to all clients.

Client RMIs
eRMI_ToServer

Sends an RMI from the client to the server.

Examples

To define a function to be implemented as RMI, use the IMPLEMENT_RMI #define from IGameObject.h.

```
#define IMPLEMENT_RMI(cls, name)
```

The following example implements a new function called Cl_SetAmmoCount in the CInventory class to be used as a client-side RMI, taking one argument of type TRMIInventory_Ammo:

```
Class CInventory : public CGameObjectExtensionHelper<CInventory, IInventory>
{
    // …
    DECLARE_CLIENT_RMI_NOATTACH(Cl_SetAmmoCount, TRMIInventory_Ammo, eNRT_ReliableOrdered);
    // …
};
```

```
IMPLEMENT_RMI(CInventory, Cl_SetAmmoCount)
{
    // Game code:
    TRMIInventory_Ammo Info(params);
    IEntityClass* pClass = gEnv->pEntitySystem->GetClassRegistry()->FindClass(Info.m_AmmoClass.c_str());
    if (pClass)
        SetAmmoCount(pClass, Info.m_iAmount);
    return true;    // Always return true - false will drop connection
}
```

The following line will invoke the function:
```
pInventory->GetGameObject()->InvokeRMI(CInventory::Cl_SetAmmoCount(),
    TRMIInventory_Ammo("Pistol", 10), eRMI_ToAllClients);
```

The following line will invoke the function:
```
pInventory->GetGameObject()->InvokeRMI(CInventory::Cl_SetAmmoCount(),
    TRMIInventory_Ammo("Pistol", 10), eRMI_ToAllClients);
```

Network Serialization and Aspects

All objects that are intended to be synchronized over the network should have a function called NetSerialize(). In the GameObject, this appears as: IGameObject::NetSerialize().

The NetSerialize() function uses a TSerialize object of type ISerialize to transform data to a stream. The serialization uses different aspects and profiles to distinguish the various types of streams.
Note
Serialized data for a given aspect and profile must remain fixed. For example, if you serialized four floats, you must always serialize four floats.

Aspects

You use aspects to logically group data together.

Aspects are defined as follows:

- `eEA_GameClient` – Information sent from the client to the server, if the client has authority over the object.
- `eEA_GameServer` – The normal server to client data stream.
- `Dynamic/Static` – Data that is constantly changing should be added to the Dynamic aspect. Objects that rarely change should be added to the Static aspect. Updates are not sent if only one value changes.
- `eEA_Script` – Used where script network data is transported, including any script RMI calls.
- `eEA_Physics` – Used where physics data is transported. It is not divided into client/server because it always uses the same path: (controlling-client) to serve other clients.

Profiles

Profiles allow an aspect's fixed format data to be different. There are potentially eight profiles per aspect, and they are only used for physics aspects (for example, switching between ragdoll and living entity).
File Access (CryPak File Archives)

Note

The Lumberyard FileIO system is now the preferred way of accessing .pak files. For more information, see Direct File Access In Lumberyard in the Amazon Lumberyard User Guide. The FileIO system can see inside .pak files transparently.

The CryPak module enables you to store game content files in a compressed or uncompressed archive.

Features

- Compatible with the standard zip format.
- Supports storing files in an archive or in the standard file system.
- Data can be read in a synchronous and asynchronous way through IStreamCallback (max 4GB offset, 4GB files).
- Files can be stored in compressed or uncompressed form.
- Uncompressed files can be read partially if required.
- File name comparison is not case sensitive.
- Supports loading of .zip or .pak files up to 4GB in size.

Unicode and Absolute Path Handling

Internally, all path-handling code is ASCII-based; as such, no Unicode (16-bit characters for different languages) functions can be used—this is to save memory and for simplicity. Because games can and should be developed with ASCII path names, no real need for Unicode exists. Game productions that don't follow these requirements have issues integrating other languages. For example, because a user might install a game to a directory with Unicode characters, absolute path names are explicitly avoided throughout the whole engine.

Layering

Usually the game content data is organized in several .pak files, which are located in the game directory. When a file is requested for an opening operation, the CryPak system loops through all registered .pak files. .pak files are searched in order of creation. This allows patch .pak files, which have been added to the build later, to be in a preferred position. It is also possible to mix .pak files with loose files, which are stored directly in the file system (not in a .pak file). If a file exists as a loose file as well as in a .pak archive, the loose file is preferred when the game is in devmode. However, to discourage cheating in the shipped game, the file stored in the .pak is preferred over the loose file when the game is not run in devmode.

Slashes

Usually forward slashes (/) are used for internal processing, but users may enter paths that contain backslashes.
Special Folder Handling

You can use the path alias @USER@ to specify a path relative to the user folder. This might be needed to store user-specific data. Windows can have restrictions on where the user can store files. For example, the program folder might not be writable at all. For that reason, screenshots, game data, and other files should be stored in the user folder. The following are examples of valid file names and paths:

@USER@/ProfilesSingle/Lisa.dat
game/Fred.dat

Internals

- A known implementation flaw exists where using more than approximately 1000 files per directory causes problems.
- Format properties:
  - The .zip file format stores each file with a small header that includes its path and filename in uncompressed text form. For faster file access, a directory is listed at the end of the file. The directory also stores the path and filename in uncompressed text form (redundant).

Creating a pak file using 7-Zip

To create a .pak file with 7-Zip's 7za.exe command line tool, use the following syntax:

7za a -tzip -r -mx0 PakFileName [file1 file2 file3 ...] [dir1 dir2 ...]

Dealing with Large Pak Files

The zip RFC specifies two types of .zip files, indicated by .zip format version 45. Old .zip files can have a 4GB offset, but if legacy I/O functions are used, it is only possible to seek +/- 2GB, which becomes the practical limit. The 4GB offsets have nothing to do with native machine types and do not change size across operating systems and compilers or configurations. The offsets for older versions of .zip files are in a machine independent uint32; the offsets for the new version .zip files are in uint64, appended to the old version structs. The version a .zip file uses is located in the header of the .zip file. Applications are free to not support the newer version. For more information, see the .ZIP File Format Specification.

Manual splits are not necessary, as RC supports auto-splitting:

- zip_sizesplit – Split .zip files automatically when the maximum configured or supported compressed size has been reached. The default limit is 2GB.
- zip_maxsize – Maximum compressed size of the .zip file in kilobytes (this gives an explicit limit).

Splitting works in all cases and supports multi-threading and incremental updates. It expands and shrinks the chain of necessary zip-parts automatically. Sorting is honored as much as possible, even in face of incremental modifications, but individual files can be appended to the end of the parts to fill in the leftover space even if this violates the sort order.

For more information about zip files, see Zip File Format Reference by Phil Katz.
Accessing Files with CryPak

Note

The Lumberyard FileIO system is now the preferred way of accessing .pak files. Direct File Access In Lumberyard. The FileIO system can see inside .pak files transparently.

In this tutorial you will learn how file reading and writing works through CryPak. The tutorial teaches you how to add new files to your project, read files from the file system and from pak archives, and write files to the file system.

Topics

• Preparation (p. 211)
• Reading Files with CryPak (p. 212)
• Writing to File System Files With CryPak (p. 214)
• Modifying Paks With CryArchive (p. 215)
• CryPak Details (p. 216)

Preparation

This tutorial demonstrates two different methods of loading a file: from inside a .pak archive, and directly from the file system. Before you can start, you need a file in a .pak archive, and a file with the same name (but with different content) in the file system. To verify which file is loaded, the example makes use of the content inside each text file.

To prepare sample files

1. Create a text file named ExampleText.txt.
2. Using a text editor, open ExampleText.txt and type in the following text:

   This sample was read from the .pak archive

3. Save the file.
4. Inside the GameSDK directory, create a subfolder called Examples.
5. Add the ExampleText.txt file to the Examples folder so that the path looks like this:

   <root>\GameSDK\Examples\ExampleText.txt

6. Run the following command from the directory root\GameSDK:

   ..\Tools\7za.exe a -tzip -r -mx0 Examples.pak Examples

   This command uses the executable file 7za.exe (located in the Tools folder) to create an archive of the Examples folder called Examples.pak. Because you ran the command from the GameSDK folder, the archive was saved to the GameSDK folder. The .pak file contains only the file Examples\ExampleText.txt.
7. Using a text editor, change the text inside the <root>\GameSDK\Examples\ExampleText.txt file to something different, for example:

   This sample was read from the file system

Now you have two different text files with the same destination path, except that one is stored directly in the file system, and the other is inside the .pak file.
Reading Files with CryPak

Now you can write some code to read the information from the ExampleText.txt file that you created.

1. Type the following, which contains the if-else statement that frames the code. The ReadFromExampleFile() function will read the contents of the file and return true if it succeeds, and false if not.

```c
char* fileContent = NULL;
if (!ReadFromExampleFile(&fileContent))
{
    CryWarning(VALIDATOR_MODULE_SYSTEM, VALIDATOR_WARNING, "ReadFromExampleFile() failed");
} else
{
    CryLogAlways("ExampleText contains %s", fileContent);
    [...] // this line will be added later on
}
```

If ReadFromExampleFile() is successful in reading ExampleText.txt, fileContent will be the space in memory that contains the text that it read.

2. Type the following, which stubs out the ReadFromExampleFile() function.

```c
bool ReadFromExampleFile(char** fileContent)
{
    CCryFile file;
    size_t fileSize = 0;
    const char* filename = "examples/exampletext.txt";
    [...] // this line will be added later on
}
```

- file of type CCryFile can make use of CryPak to access files directly from the file system or from inside a .pak archive.
- fileSize - Defines the end of the message. In this case, reading does not end by detecting the null character '\0'.
- filename - Specifies the path of the file to be loaded and is case-insensitive.

3. Type the following, which uses CryPak to search the file.

```c
char str[1024];
if (!file.Open(filename, "r"))
{
    sprintf(str, "Can't open file, (%s)", filename);
    CryWarning(VALIDATOR_MODULE_SYSTEM, VALIDATOR_WARNING, "%s", str);
    return false;
}
```

- Open() invokes CryPak to search the file specified by filename.
- File access mode "r" specifies that a plain text file is going to be read. To read a binary file, use "rb" instead.

4. Type the following, which gets the length of the file. If the file is not empty, it allocates the memory required as indicated by the file length. It then reads the file content. It aborts if the size of the content is not equal to the file length.

```c
fileSize = file.GetLength();
```
if (fileSize <= 0) {
    sprintf(str, "File is empty, (%s)", filename);
    CryWarning(VALIDATOR_MODULE_SYSTEM, VALIDATOR_WARNING, "%s", str);
    return false;
}

char* content = new char[fileSize + 1];
content[fileSize] = '\0';

if (file.ReadRaw(content, fileSize) != fileSize) {
    delete[] content;
    sprintf(str, "Can't read file, (%s)", filename);
    CryWarning(VALIDATOR_MODULE_SYSTEM, VALIDATOR_WARNING, "%s", str);
    return false;
}

• content is the local pointer to a char array in memory which gets initialized by the length returned by GetLength() and an extra null character.

• ReadRaw fills content with the information read from the text file. In case of a failure, the allocated memory of content is freed.

5. Type the following, which closes the file handle and sets the fileContent pointer so that the locally created data can be used outside the function. Finally, it returns true since the reading was successful.

```cpp
file.Close();

*fileContent = content;
return true;
```

Note
In the example, the caller of ReadFromExampleFile() is responsible for freeing the heap memory which has been allocated to store the data from the text file. Thus, after the data has been used, be sure to add the call `delete[] fileContent;`.

6. To check if the reading was successful, run the game and check the Game.log file.

Complete example code (file reading)

Calling ReadFromExampleFile()

```cpp
char* fileContent = NULL;
if (!ReadFromExampleFile(&fileContent)) {
    CryWarning(VALIDATOR_MODULE_SYSTEM, VALIDATOR_WARNING, "ReadFromExampleFile() failed");
} else {
    CryLogAlways("ExampleText contains %s", fileContent);
    delete[] fileContent;
}
```

ReadFromExampleFile() implementation

```cpp
bool ReadFromExampleFile(char** fileContent) {
```
Writing to File System Files With CryPak

Writing a file is similar to the process for reading one. To write to files, you use `CCryFile::Write`, which always writes to the file system and never to `.pak` archives. For information on writing files to archive files, see Modifying Paks With CryArchive (p. 215).

1. Type the following, which contains the `if-else` statement that frames the code for writing to a file. The `WriteToExampleFile()` function write will write the contents of the file and return `true` if it succeeds, and `false` if not.

```cpp
char* newContent = "File has been modified";
bool appendToFile = false;
if (!WriteToExampleFile(newContent, strlen(newContent), appendToFile))
{
    CryWarning(VALIDATOR_MODULE_SYSTEM, VALIDATOR_WARNING, "WriteToExampleFile() failed");
}else
{
    CryLogAlways("Text has been written to file, %s", newContent);
}
```

- `WriteToExampleFile()` takes the following three parameters:
  - `newContent` - The text which will be written to ExampleText.txt on the file system.
• strlen(newContent) - Returns size of newContent, which is the number of bytes to be written.
• appendToFile - true if newContent will be added to the already existing content; false if the file will be overwritten.

2. Type the following for the WriteToExampleFile() function.

```cpp
bool WriteToExampleFile(char* text, int bytes, bool appendToFile)
{
    CCryFile file;
    const char* filename = "examples/exampletext.txt";

    assert(bytes > 0);
    char* mode = NULL;
    if (appendToFile)
        mode = "a";
    else
        mode = "w";

    char str[1024];
    if (!file.Open(filename, mode))
    {
        sprintf(str, "Can't open file, (%s)", filename);
        CryWarning(VALIDATOR_MODULE_SYSTEM, VALIDATOR_WARNING, "%s", str);
        return false;
    }
    [...]
    file.Close();
    return true;
}
```

• mode specifies if the text is to be appended to the existing file or if it will overwrite existing file contents. "w" means 'write' to a clean file, and "a" means 'append' to the existing file.

3. The final step writes the text to the file and returns the number of bytes written, or an error message if none were written.

```cpp
int bytesWritten = file.Write(text, bytes);
assert(bytesWritten == bytes);

if (bytesWritten == 0)
{
    sprintf(str, "Can't write to file, (%s)", filename);
    CryWarning(VALIDATOR_MODULE_SYSTEM, VALIDATOR_WARNING, "%s", str);
    return false;
}
```

• bytesWritten tells how many bytes were written by calling the Write() function.

# Modifying Paks With CryArchive

This section contains a short example that shows how files are added, updated and removed from an archive. The example intentionally uses the USER folder instead of the GameSDK folder because the .pak files inside the GameSDK folder are loaded by default at startup and therefore are marked as Read-Only. (Files in the USER folder are not loaded by default at startup.)

```cpp
string pakFilename = PathUtil::AddSlash("@USER@") + "Examples.pak";
const char* filename = "Examples/ExampleText.txt";
```
char* text = "File has been modified by CryArchive";
unsigned length = strlen(text);
_smart_ptr<ICryArchive> pCryArchive = gEnv->pCryPak->OpenArchive(pakFilename.c_str(),
ICryArchive::FLAGS_RELATIVE_PATHS_ONLY | ICryArchive::FLAGS_CREATE_NEW);
if (pCryArchive)
{
pCryArchive->UpdateFile(filename, text, length, ICryArchive::METHOD_STORE, 0);
}

• UpdateFile() - Modifies an existing file inside the .pak archive or creates a new one if it does not exist.
• ICryArchive::FLAGS_CREATE_NEW - Forces a new .pak file to be created. If you want to add (append) files, remove this flag.
• To remove files or folders from an archive, use one of the following commands in place of UpdateFile(): RemoveFile(), RemoveDir() or RemoveAll().

CryPak Details

Initialization

To ensure that .pak files can be accessed from game code at anytime, the CrySystem module initializes CryPak in CSystem::Init by calling the following functions:

• InitFileSystem(startupParams.pGameStartup);
• InitFileSystem_LoadEngineFolders();

Tip
A good spot to test game initialization is inside Game.cpp at the beginning of CGame::Init.

Pak file type priorities

Whether CryPak processes files in the file system first, or files in .pak files first, depends on the value of pakPriority. The default value of pakPriority depends on the configuration settings of your build, but it can also manually be changed by assigning the console variable sys_PakPriority the values 0, 1, 2 or 3. The meaning of these values is show in the enum EPakPriority:

PakVars.h

enum EPakPriority
{
    ePakPriorityFileFirst = 0,
    ePakPriorityPakFirst  = 1,
    ePakPriorityPakOnly   = 2,
    ePakPriorityFileFirstModsOnly = 3,
};

Pak loading and search priorities

The reason for adding the new pak file to the GameSDK folder in this example is because .pak files are loaded from the GameSDK path first. The loading order and search order of .pak file folders are as follows. Note that the loading order and the search order are the reverse of each other.

.pak file load order
1. GameSDK: `<root>\GameSDK\*.pak`
2. Engine: `<root>\Engine\`
   a. Engine.pak
   b. ShaderCache.pak
   c. ShaderCacheStartup.pak
   d. Shaders.pak
   e. ShadersBin.pak
3. Mods: root\Mods\MyMod\GameSDK\*.pak (this assumes that you run the game with the command argument -mod "MyMod")

.pak file search order

1. Mods If more than one mod folder exists, they will be checked in the reverse order in which they were added.
2. Engine
3. GameSDK
Flow Graph System

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard’s new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

Flow Graph is a visual scripting system that allows you to implement complex game logic without having to touch any code. Complex logic can be created with only a few clicks and an extensive library of nodes provides everything needed to fully control entities and AI agents in a level.

Flow Graph can also be used to prototype gameplay, effects, and sound design, with a level containing multiple flow graphs performing different tasks at the same time.

Flow graphs consist of nodes and links. Nodes can represent level entities (entity node) or actions (component node) that may perform a specific action on a target entity. Links are used to connect nodes, and are represented as lines that connect the inputs and outputs between nodes.

Flow Graph logic is stored in XML files and can be exported for use in other levels. As a flow graph is associated with a specific entity, the graph is always exported along with the entity. Layers are supported.

Topics
- Using Flow Graph Editor (p. 218)
- Flow Graph Scripts (p. 220)
- Managing Flow Graphs (p. 221)
- Using Flow Graph Nodes (p. 223)
- Creating Flow Graph Nodes (p. 226)
- Flow Graph Node Reference (p. 233)
- Using Flow Graph Links (p. 688)
- Using Flow Graph Tokens (p. 689)
- Managing Flow Graph Modules (p. 690)
- Debugging Flow Graph (p. 692)
- Placing Cached Shadows (p. 693)
- Cloud Canvas Flow Graph Node Reference (p. 695)

Using Flow Graph Editor

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard’s new visual scripting environment.
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 about the latest features, see the Amazon Lumberyard User Guide.

Flow Graph editor uses drag-and-drop modules and connection links to various inputs and outputs to perform visual scripting. The following shows the components of the Flow Graph editor:

- **Node graph** - main window grid for displaying flow graph nodes and connections
- **Components** - browser tree pane for nodes
- **Graphs** - browser tree pane for graphs and entities
- **Properties** - pane for showing node input and output properties
- **Search** - pane for searching graphs and nodes
- **SearchResults** - pane for displaying search results
- **Breakpoints** - pane for displaying breakpoints
Flow Graph Scripts

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.
Flow Graph scripts are organized into four different categories, and contained in the **Graphs** folder tree in the Flow Graph Editor.

**Level Flowgraphs**

This directory contains script files that are specific to the level that is currently open, and is organized as follows:

- **Entities** – Entity files are the flow graphs created and associated with an entity that has been placed in the level.
- **Components** – Component files are the flow graphs created and associated with a component that has been placed in the level.
- **Modules** – Modules that are specific to the level that is currently open.

**Global Flowgraphs**

- **UI Actions** - Used to encapsulate UI logic for easy debugging and maintenance.

**Flow Graph Prefabs**

Using Flow Graph, you can communicate directly to and from a prefab instance just like an entity by using prefab events. Simply create an event inside a prefab, give it a name, and then reference the prefab instance as you normally do for an entity.

**External Files**

These are Flow Graph scripts that have been imported.

**Managing Flow Graphs**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about **Script Canvas**, Lumberyard’s new visual scripting environment.

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 about the latest features, see the **Amazon Lumberyard User Guide**.

Each flow graph is associated with a specific entity and is stored as a property of the entity. The name of the flow graph is the name of the entity for which it has been created. When the entity name is changed, the name in the flow graph is also automatically changed. When the entity is saved or exported, the flow graph belonging to it is also automatically saved.

There are two types of flow graphs: global flow graphs, which are used in multiple levels, and level flow graphs, which are associated with a single level.
To create a flow graph for an entity

1. In Rollup Bar, on the Objects tab, for an entity previously created, under Flow Graph, click Create.
2. Alternatively, right-click the entity in the viewport, then click Create Flow Graph. If this is the first flow graph in a level, you need to select a group to place the flow graph with, or click New to create a new group name for the flow graph. The Flow Graphs window displays the new flow graph in the tree.

To manage flow graphs

- In Flow Graph Editor, right-click the applicable flow graph in the Flow Graphs tree, then do the following as needed:
  - To delete a flow graph, click Delete Graph.
    
    **Note**
    When an entity is deleted from a level, the associated flow graph is also deleted.
  - To enable or disable a flow graph, toggle Enable or Disable.
  - To enable or disable all flow graphs in a group, right-click the parent folder, then click Enable All or Disable All as needed. A disabled flow graph is displayed as crossed out, which means that all nodes in the flow graph are ignored when the game is running.
  - To move a flow graph to another group, right-click the parent folder, click RenameFolder/MoveGraphs, then select a group from the list or click New to move it to its own new group and name it.

When a level is exported with some flow graphs disabled, their disabled state is also exported to the game.

**Saving Flow Graphs**

The method of saving flow graphs differ depending on whether it is a global flow graph or a level flow graph.

Global flow graphs, which are listed under Graphs, Global, are saved by selecting the flow graph and then clicking File, Save.
Level flow graphs, which are listed under **Graphs, Level**, are saved automatically when either the level they are in is saved or the layer that they are on is saved. A layer gets saved whenever the corresponding level is saved.

### Grouping Flow Graphs

#### To create a flow graph group

1. In the graph pane, select two or more flow graph nodes by CTRL+ click on each one.
2. Right-click the graph pane, and click **Group**. A box appears around the nodes.
3. Type a name for the group.

You can rename, move, add to, and remove a group.

#### To manage flow graph groups

1. To rename a group, double-click the group's name and type a new name.
2. To collapse a group to save space, click the down-arrow icon for the group. To expand the group back, click on the icon again.
3. To move a node within a group, click on the node's title bar and drag it to the desired location.
4. To move a group, click on an empty space in the group and drag it to the desired location.
5. To add a node to a group, click to select the group, Ctrl+click on the applicable node, right-click the graph pane, then click **Add group**. The group's box now encloses the new node.
6. To remove a node from a group, click to select it, right-click on an empty space in the group, then click **Ungroup**. The nodes selected are removed from the group. If the group as a whole is selected, the group is removed entirely.
7. To remove a group entirely, right-click the group's name and click in the **Ungroup**.

### Importing and Exporting Flow Graphs

Flow graphs are saved as XML files and can be exported and imported.

#### To export a flow graph

- Select the nodes for export by CTRL+click each node, then right-click the final node, click **Selection, Export Selected Nodes**, then enter a file name for it.

You can import a previously exported flow graph's nodes into another flow graph as follows:

#### To import a flow graph

- Open the target flow graph you want to import to add the exported flow graph nodes to, right-click anywhere in the graph pane, click **Import**, then enter the name of file you want to import.

The imported flow graph is positioned relative to the old flow graph.

### Using Flow Graph Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about **Script Canvas**, Lumberyard's new visual scripting environment.
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 about the latest features, see the Amazon Lumberyard User Guide.

Nodes can represent level entities (entity node) or actions (component node) that may perform a specific action on a target entity. A node is represented in Flow Graph as a box with inputs and outputs.

Node Input/Output Ports

A node consists of input ports on the left side for receiving information and output ports on the right side for transmitting information. Output ports are activated depending on the function of the node. Ports can have the following different data types.

Node Port Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Color</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any</td>
<td>n/a</td>
<td>Unspecified, any data type can be received</td>
</tr>
<tr>
<td>Boolean</td>
<td>Blue</td>
<td>True or false value</td>
</tr>
<tr>
<td>EntityID</td>
<td>Green/Red</td>
<td>Value that uniquely identifies any entity in a level</td>
</tr>
<tr>
<td>Float</td>
<td>White</td>
<td>Floating-point 32-bit value</td>
</tr>
<tr>
<td>Integer</td>
<td>Red</td>
<td>Positive or negative 32-bit number</td>
</tr>
<tr>
<td>UInt64</td>
<td>n/a</td>
<td>Positive or negative 64-bit number</td>
</tr>
<tr>
<td>String</td>
<td>Turquoise</td>
<td>Array of characters used for storing text</td>
</tr>
<tr>
<td>Vec3</td>
<td>Yellow</td>
<td>3D vector consisting of three floating-point values. Used for storing positions, angles, or color values</td>
</tr>
</tbody>
</table>
### Adding Entity Nodes

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Color</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Void</td>
<td>Green</td>
<td>Used for ports that do not accept any value but are instead triggered to pass the flow of control through a flow graph.</td>
</tr>
</tbody>
</table>

Differing colors for node backgrounds and links indicate the following:

- Nodes with a red background and a yellow title bar are debugging nodes and are not functional in release builds.
- Links that connect debugging nodes are yellow.
- Dotted links indicate they are disabled (by right-clicking them)

Values whose data type don't match the input port data type are automatically converted to match the type of the port connected to, if possible. Any output port can be connected to any input port, no matter what data type. An integer with the value `1` can be fed in a Boolean input port and converted to a `True` value to match the data type of the port. For some component nodes, there is an input port at the top of the entity that is used for setting the target entity of the node.

**Note**

Mixing node port types or data types can result in unexpected behavior. For example while a `Math:SetColor` node input port is a `Vec3` data type, it treats input from a `Vec3:SetVec3` node differently than from a `Math:SetColor` node, both of which output a `Vec3` data type. While the port types for both nodes are vector, the `Vec3:SetVec3` are a group of three floating-point values whereas the `Math:SetColor` data type are a group of colors that range from 0-255.

### Adding Entity Nodes

Entity nodes require that a level entity first be selected. To add an entity node, select an entity and open the graph where you want to add the entity. Next, open the graph context menu by right-clicking the main editing pane.

**To add an Entity node**

1. In the left-side **Flow Graphs** tree, expand **Entities\fg** and select the applicable entity.
2. Right-click anywhere in the graph pane and click **Add Selected Entity**.
3. Or, right-click anywhere in the graph pane and click **Add Graph Default Entity**, which always adds the entity to the flow graph to which it is attached.

### Adding Component Nodes

Component nodes can be added from within the graph and don't require any selected entity. There are three ways to add these nodes, the context menu, the component node list window and the **QuickSearchNode** (keyboard shortcut: Q).

To add a new component node, open the context menu by right-clicking the main editing pane, and then select Add Node. A long list of sub-folders are displayed, and a node can be selected from any directory. Select Entity to open the folder with the entity-related component nodes. Select EntityPos to complete the procedure.

**To add a Component node**

- Right-click anywhere in the graph pane, click **Add Node**, and select a node from the list.
Managing Nodes

You can easily move, copy, edit, and delete Flow Graph nodes as follows. All links between selected nodes are also moved when the nodes are moved and automatically rearrange themselves.

**To move a node**

1. Click and drag the node on the graph pane. Multiple nodes can be moved by holding down the Ctrl key and clicking the applicable nodes.
2. Or, use the mouse to draw a box around all the applicable nodes that need to be moved.

**To copy a node**

1. Right-click the node, click *Copy*, then click *Paste* at the desired location in the graph pane. Click *Paste With Links* to also copy all connected links.
2. Or, click the node, press Ctrl+C, then press Ctrl+V at the desired location.

**To edit a node**

There are two ways to edit a node's properties.

1. Double-click the applicable node input and change the property.
2. Or, change the property as listed under *Inputs* in the right-side panel of Flow Graph Editor.

**To delete a node**

There are two ways to delete a node. Once a node has been deleted, all the connected links are also automatically removed.

1. Right-click the node and click *Delete*.
2. Or, click the node and press the keyboard Delete key.

Creating Flow Graph Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard’s new visual scripting environment.

You can use a .cpp file to create new flow graph nodes. For multiple flow graph nodes that will belong to the same group, use a single .cpp file. Headers aren’t needed except for some specialized nodes.

Use the following code template for your .cpp file and save the file to the dev\Code\CryEngine\CryAction\FlowSystem\Nodes directory.

In the template you can choose between an eNCT_Instanced node and a eNCT_Singleton. A singleton node creates one instance with a small memory footprint, although you can still use multiple
nodes in your flow graph. Use singleton whenever you are not saving state data such as member variables.

```cpp
#include "StdAfx.h"
#include "FlowBaseNode.h"

class CFlowNode_your_flow_node_name : public CFlowBaseNode<eNCT_Instanced> {
public:
    CFlowNode_your_flow_node_name(SActivationInfo* pActInfo) {
    }

    virtual IFlowNodePtr Clone(SActivationInfo *pActInfo) {
        return new CFlowNode_your_flow_node_name(pActInfo);
    }

    virtual void GetMemoryUsage(ICrySizer* s) const {
        s->Add(*this);
    }

    virtual void GetConfiguration(SFlowNodeConfig& config) {
        static const SInputPortConfig in_config[] = {
            {0}
        };
        static const SOutputPortConfig out_config[] = {
            {0}
        };
        config.sDescription = _HELP("your_flow_node_toolip_description");
        config.pInputPorts = in_config;
        config.pOutputPorts = out_config;
        config.SetCategory(EFLN_APPROVED);
    }

    virtual void ProcessEvent(EFlowEvent event, SActivationInfo* pActInfo) {
        switch (event) {
            
        }
    }
};
REGISTER_FLOW_NODE("your_flow_node_group:your_flow_node_name",
CFlowNode_your_flow_node_name);
```

For your flow node group, create a corresponding subfolder in the Flow Graph editor node selector where this node will be placed in the hierarchy.

### Output Ports

You can add an output port by modifying the GetConfiguration function as shown in the following example:

```cpp
class CFlowNode_your_flow_node_name : public CFlowBaseNode<eNCT_Instanced> {
public:
    // ...

    virtual void GetConfiguration(SFlowNodeConfig& config) {
```
static const SInputPortConfig in_config[] = {
    {0}
};
static const SOutputPortConfig out_config[] = {
    OutputPortConfig<int>("your_output", _HELP("your_help_text")),
    {0}
};
config.sDescription = _HELP("your_flow_node_tooltip_description");
config.pInputPorts = in_config;
config.pOutputPorts = out_config;
config.nFlags = 0;
}

Input Ports

You can add an input port by modifying the GetConfiguration function as shown in the following example:

class CFLOW_NODE_your_flow_node_name : public CFLOW_BASE_NODE<eNCT_Instance>
{
    public:
    // ...

    virtual void GetConfiguration(SFLOW_NODE_CONFIG& config)
    {
        static const SInputPortConfig in_config[] = {
            InputPortConfig<int>("your_input", _HELP("your_help_text")),
            {0}
        };
        static const SOutputPortConfig out_config[] = {
            {0}
        };
        config.sDescription = _HELP("your_flow_node_tooltip_description");
        config.pInputPorts = in_config;
        config.pOutputPorts = out_config;
        config.nFlags = 0;
    }

OutputPortConfig is a helper function that is useful for filling a small structure with appropriate data.

Available data types for this function include SFLOW_SYSTEM_VOID, Int, Float, EntityId, Vec3, String, and Bool. SFLOW_SYSTEM_VOID is a special data type that represents "no value".

OutputPortConfig takes the following parameters:

- Port name that is used internally and for saving the flow graph. Do not change this parameter later as doing so will break script compatibility for all flow graphs that use this node.
  
  **Note**
  
  Do not use the underscore "_" character as this was used in previous versions to specify a specialized editor for the port.

- Description used to display tooltip help text on mouse hover in the Flow Graph editor.

- Human-readable name used to display the name of the port in the Flow Graph editor. This is used to visually override a port name without breaking script compatibility.

To emit a value from the output port, use the function

CFLOW_BASE_NODE::ActivateOutput(pActInfo, nPort, value). This function takes a pActInfo, which is typically passed to ProcessEvent(), the nPort port identifier (count starts at zero from the top of out_config), and a value of the same type as the port.
InputPortConfig is a helper function that is useful for filling a small structure with appropriate data. Available data types for this function include SFlowSystemVoid, Int, Float, EntityId, Vec3, String, and Bool. SFlowSystemVoid is a special data type that represents "no value".

InputPortConfig takes the following parameters:

- Port name used internally and for saving the flow graph. Do not change this parameter later as doing so will break script compatibility for all flow graphs that use this node.
  
  **Note**
  Do not use the underscore "_" character as this was used in previous versions to specify a specialized editor for the port.

- Default value of the port when a new node is created.

- Description used to display tooltip help text on mouse hover in the Flow Graph editor.

- Human-readable name used to display the name of the port in the Flow Graph editor. Use to visually override a port name without breaking script compatibility.

- Formatted string that specifies how the UI should function when setting the port value. You can choose a specialized widget or modify the allowed value range of the input.

### Input Port UI Configuration

You can define the interface for setting the input port value by passing a series of options in the form of a string with key–value pairs in InputPortConfig.

**Setting the input value range**

This will limit the widget's arrows and ramp and clamp manually-inserted values as shown in the figure:

```
_UICONFIG("v_min=0, v_max=10")
```

**Setting the Drop-down List**

There are several types of enums that you can use to display a drop-down list of readable strings. Each string maps to a value that is used by the node and that persists when the flow graph is saved. Enums can be of type `int` or `float` as shown in the following figure and code example.

```
_UICONFIG("enum_int:Relaxed=0,Alert=1,Combat=2,Crouch=3")
```

An enum can also be of type `string` with or without mapping to another value:

```
_UICONFIG("enum_string:a,b,c")
_UICONFIG("enum_string:DisplayA=a,DisplayB=b,DisplayC=c")
```

Enums can also refer to the global and dynamic UI enums defined in InitUIEnums.
Optionally, the enum can depend on another port to affect the available selection:

\[
\begin{align*}
&\text{\_UICONFIG("enum\_global:ENUM\_NAME"}) \\
&\text{\_UICONFIG("enum\_global:vehicleLightTypes")} \\
&\text{\_UICONFIG("enum\_global\_def:ENUM\_NAME")} \\
&\text{\_UICONFIG("enum\_global\_ref:ENUM\_NAME\_FORMAT\_STRING:REF\_PORT")}
\end{align*}
\]

### Setting a Specialized Property Editor

You can indicate a dedicated property editor with the `dt` keyword followed by parameters optionally needed by the editor as shown in the following code example:

\[
\begin{align*}
&\text{\_UICONFIG("dt=editorName") \_UICONFIG("dt=entityProperties, ref\_entity=entityId")} \\
&\text{\_UICONFIG("dt=matparamslot, slot\_ref=Slot, sub\_ref=SubMtlId, param=float")}
\end{align*}
\]

There is a set of available editors that can be referenced in the following table:

<table>
<thead>
<tr>
<th>Editor Name</th>
<th>Editor Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>snd</td>
<td>IVariable::DT_SOUND</td>
</tr>
<tr>
<td>sound</td>
<td>IVariable::DT_SOUND</td>
</tr>
<tr>
<td>clr</td>
<td>IVariable::DT_COLOR</td>
</tr>
<tr>
<td>color</td>
<td>IVariable::DT_COLOR</td>
</tr>
<tr>
<td>tex</td>
<td>IVariable::DT_TEXTURE</td>
</tr>
<tr>
<td>texture</td>
<td>IVariable::DT_TEXTURE</td>
</tr>
<tr>
<td>obj</td>
<td>IVariable::DT_OBJECT</td>
</tr>
<tr>
<td>object</td>
<td>IVariable::DT_OBJECT</td>
</tr>
<tr>
<td>file</td>
<td>IVariable::DT_FILE</td>
</tr>
<tr>
<td>text</td>
<td>IVariable::DT_LOCAL_STRING</td>
</tr>
<tr>
<td>equip</td>
<td>IVariable::DT_EQUIP</td>
</tr>
<tr>
<td>reverbpreset</td>
<td>IVariable::DT_REVERBPRESET</td>
</tr>
<tr>
<td>aianchor</td>
<td>IVariable::DT_AI_ANCHOR</td>
</tr>
<tr>
<td>aibehavior</td>
<td>IVariable::DT_AI_BEHAVIOR</td>
</tr>
<tr>
<td>aicharacter</td>
<td>IVariable::DT_AI_CHARACTER</td>
</tr>
<tr>
<td>aipfpropertieslist</td>
<td>IVariable::DT_AI_PFPROPERTIESLIST</td>
</tr>
<tr>
<td>aientityclasses</td>
<td>IVariable::DT_AIENTITYCLASSES</td>
</tr>
<tr>
<td>soclass</td>
<td>IVariable::DT_SOCLASS</td>
</tr>
<tr>
<td>soclasses</td>
<td>IVariable::DT_SOCLASSES</td>
</tr>
<tr>
<td>sostate</td>
<td>IVariable::DT_SOSTATE</td>
</tr>
<tr>
<td>sostates</td>
<td>IVariable::DT_SOSTATES</td>
</tr>
<tr>
<td>Editor Name</td>
<td>Editor Type</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>sopattern</td>
<td><code>IVariable::DT_SOSTATEPATTERN</code></td>
</tr>
<tr>
<td>soaction</td>
<td><code>IVariable::DT_SOACTION</code></td>
</tr>
<tr>
<td>sohelper</td>
<td><code>IVariable::DT_SOHELPER</code></td>
</tr>
<tr>
<td>sonavhelper</td>
<td><code>IVariable::DT_SONAVHELPER</code></td>
</tr>
<tr>
<td>soanimhelper</td>
<td><code>IVariable::DT_SOANIMHELPER</code></td>
</tr>
<tr>
<td>soevent</td>
<td><code>IVariable::DT_SOEVENT</code></td>
</tr>
<tr>
<td>customaction</td>
<td><code>IVariable::DT_CUSTOMACTION</code></td>
</tr>
<tr>
<td>gametoken</td>
<td><code>IVariable::DT_GAMETOKEN</code></td>
</tr>
<tr>
<td>mat</td>
<td><code>IVariable::DT_MATERIAL</code></td>
</tr>
<tr>
<td>seq</td>
<td><code>IVariable::DT_SEQUENCE</code></td>
</tr>
<tr>
<td>mission</td>
<td><code>IVariable::DT_MISSIONOBJ</code></td>
</tr>
<tr>
<td>anim</td>
<td><code>IVariable::DT_USERITEMCB</code></td>
</tr>
<tr>
<td>animstate</td>
<td><code>IVariable::DT_USERITEMCB</code></td>
</tr>
<tr>
<td>animstateEx</td>
<td><code>IVariable::DT_USERITEMCB</code></td>
</tr>
<tr>
<td>bone</td>
<td><code>IVariable::DT_USERITEMCB</code></td>
</tr>
<tr>
<td>attachment</td>
<td><code>IVariable::DT_USERITEMCB</code></td>
</tr>
<tr>
<td>dialog</td>
<td><code>IVariable::DT_USERITEMCB</code></td>
</tr>
<tr>
<td>matparamslot</td>
<td><code>IVariable::DT_USERITEMCB</code></td>
</tr>
<tr>
<td>matparamname</td>
<td><code>IVariable::DT_USERITEMCB</code></td>
</tr>
<tr>
<td>matparamcharatt</td>
<td><code>IVariable::DT_USERITEMCB</code></td>
</tr>
<tr>
<td>seqid</td>
<td><code>IVariable::DT_SEQUENCE_ID</code></td>
</tr>
<tr>
<td>lightanimation</td>
<td><code>IVariable::DT_LIGHT_ANIMATION</code></td>
</tr>
<tr>
<td>formation</td>
<td><code>IVariable::DT_USERITEMCB</code></td>
</tr>
<tr>
<td>communicationVariable</td>
<td><code>IVariable::DT_USERITEMCB</code></td>
</tr>
<tr>
<td>uiElements</td>
<td><code>IVariable::DT_USERITEMCB</code></td>
</tr>
<tr>
<td>uiActions</td>
<td><code>IVariable::DT_USERITEMCB</code></td>
</tr>
<tr>
<td>uiVariables</td>
<td><code>IVariable::DT_USERITEMCB</code></td>
</tr>
<tr>
<td>uiArrays</td>
<td><code>IVariable::DT_USERITEMCB</code></td>
</tr>
<tr>
<td>uiMovieclips</td>
<td><code>IVariable::DT_USERITEMCB</code></td>
</tr>
<tr>
<td>uiVariablesTmpl</td>
<td><code>IVariable::DT_USERITEMCB</code></td>
</tr>
<tr>
<td>uiArraysTmpl</td>
<td><code>IVariable::DT_USERITEMCB</code></td>
</tr>
</tbody>
</table>
Trigger Ports

It can be useful to have a trigger signal as an input or output port. You can implement these ports using the `Input/OutputPortConfig_Void` or `Input/OutputPortConfig_AnyType` data types. Do not use the `Boolean` data type.

Update Event

If you want an update loop for your node instead of having it react on ports, you can use the following code to add your node to the list of regularly updated nodes. You can also choose to enable the update event temporarily.

The following code adds your node to the list of regularly updated nodes:

```cpp
pActInfo->pGraph->SetRegularlyUpdated( pActInfo->myID, true);
```

You will get a single `ProcessEvent(eFE_Updated)` call per game update call.

To remove it from this list, call the same function with `false` as the second parameter.
Flow Graph Node Reference

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

This section provides a listing of the flow graph nodes, including the various types of nodes, input and output ports, and their uses. The most commonly-used and important nodes include Entity (and ComponentEntity), Interpolate, Logic, Math, Mission, Time, Vec3, and Debug nodes.

For a list of UI, VR and Cloud Canvas flow graph nodes, see UI Flow Graph Nodes (p. 482), Setting Up Virtual Reality with Flow Graph, and Cloud Canvas Flow Graph Node Reference (p. 695).

**Note**
Node input/output port descriptions are also available as tool tip text when you pause on a port in the node graph or in the Properties pane in the Flow Graph editor.

**Topics**
- Actor Nodes (p. 234)
- AI Nodes (p. 239)
- AISequence Nodes (p. 260)
- Animations Nodes (p. 267)
- Audio Nodes (p. 276)
- Camera Nodes (p. 281)
- ComponentEntity Nodes (p. 283)
- CustomAction Nodes (p. 292)
- Debug Nodes (p. 296)
- Dialog Nodes (p. 308)
- Dynamic Response Nodes (p. 310)
- Engine Nodes (p. 313)
- Entity Nodes (p. 315)
- Environment Nodes (p. 329)
- FeatureTest Nodes (p. 335)
- Force Feedback Nodes (p. 337)
- Game Nodes (p. 340)
- Helicopter Nodes (p. 346)
- Image Nodes (p. 348)
- Input Nodes (p. 360)
- Interpolate Nodes (p. 379)
- Intersection Tests Nodes (p. 385)
- Iterator Nodes (p. 386)
Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to configure various actor behaviors and settings.

**Note**
These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at Lumberyard Downloads.

Topics

- JSON Nodes (p. 389)
- Kinect Nodes (p. 392)
- Logic Nodes (p. 393)
- Material Nodes (p. 404)
- MaterialFX Nodes (p. 407)
- Math Nodes (p. 409)
- Mission Nodes (p. 427)
- Module Nodes (p. 432)
- Movement Nodes (p. 436)
- Physics Nodes (p. 439)
- Prefab Nodes (p. 445)
- ProceduralMaterial Nodes (p. 446)
- Stereo Nodes (p. 458)
- String Nodes (p. 459)
- System Nodes (p. 463)
- Time Nodes (p. 465)
- Twitch Nodes (p. 474)
- UI Flow Graph Nodes (p. 482)
- Vec3 Nodes (p. 646)
- Vehicle Nodes (p. 654)
- Video Nodes (p. 664)
- VideoPlayback Nodes (p. 665)
- Virtual Reality (VR) Nodes (p. 668)
- Weapon Nodes (p. 672)
- XML Nodes (p. 675)
• Damage node (p. 235)
• EnslaveCharacter node (p. 235)
• GrabObject node (p. 236)
• HealthCheck node (p. 236)
• HealthGet node (p. 237)
• HealthSet node (p. 237)
• LocalPlayer node (p. 238)
• PlayMannequinFragment node (p. 238)
• ProcClipEventListener node (p. 239)

**Damage node**

Used to damage the chosen entity using the **Damage** input value when the **Trigger** input is activated.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Damage</td>
<td>Integer</td>
<td>Type of damage to inflict</td>
</tr>
<tr>
<td>DamageRelative</td>
<td>Integer</td>
<td>Level of relative damage</td>
</tr>
<tr>
<td>Position</td>
<td>Vec3</td>
<td>Location where the damage occurs</td>
</tr>
</tbody>
</table>

**EnslaveCharacter node**

Used to enslave one character to another character.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enslave</td>
<td>Any</td>
<td>Enslaves the character</td>
</tr>
<tr>
<td>Unenslave</td>
<td>Any</td>
<td>Frees the character</td>
</tr>
<tr>
<td>Slave</td>
<td>Any</td>
<td>Character to enslave</td>
</tr>
<tr>
<td>ScopeContext</td>
<td>String</td>
<td>Context of the scope</td>
</tr>
<tr>
<td>DB</td>
<td>String</td>
<td>Optional database name</td>
</tr>
</tbody>
</table>
### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Tigger for successful enslaving</td>
</tr>
<tr>
<td>Failed</td>
<td>Any</td>
<td>Trigger for failed enslaving</td>
</tr>
</tbody>
</table>

### GrabObject node

Used by the chosen entity to grab an object, then to drop or throw the object.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Any</td>
<td>ID of the object to grab</td>
</tr>
<tr>
<td>grab</td>
<td>Any</td>
<td>Grabs the object</td>
</tr>
<tr>
<td>drop</td>
<td>Any</td>
<td>Drops the object</td>
</tr>
<tr>
<td>throw</td>
<td>Boolean</td>
<td>Throws the object</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
<td>Boolean</td>
<td>True if the object was successfully dropped or thrown</td>
</tr>
<tr>
<td>grabbedObjId</td>
<td>Any</td>
<td>ID of the grabbed object</td>
</tr>
</tbody>
</table>

### HealthCheck node

Used to check the health of the chosen actor entity. When the node is triggered the health of the entity is checked and if it is within the defined MinHealth and MaxHealth values, a True will be output on the InRange port.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Activates the port</td>
</tr>
<tr>
<td>MinHealth</td>
<td>Float</td>
<td>Lower limit of health range</td>
</tr>
</tbody>
</table>
Lumberyard Legacy Reference
Actor Nodes

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxHealth</td>
<td>Float</td>
<td>Upper limit of health range</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InRange</td>
<td>Boolean</td>
<td>True if health is between the MinHealth and MaxHealth values</td>
</tr>
</tbody>
</table>

HealthGet node

Used to get the health of an actor entity.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Activate this port to get the current health of the chosen entity</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Health</td>
<td>Integer</td>
<td>Current health of the chosen entity</td>
</tr>
</tbody>
</table>

HealthSet node

Used to set the health of the actor entity.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Activate this port to set the current health of the chosen entity</td>
</tr>
<tr>
<td>Value</td>
<td>Float</td>
<td>Health value to the set for the chosen entity</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Health</td>
<td>Integer</td>
<td>Current health of the chosen entity</td>
</tr>
</tbody>
</table>
LocalPlayer node

Used to update and output the ID of the local player entity.

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>update</td>
<td>Any</td>
<td>Updates the ID of the local player entity; required for multiplayer games</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>Any</td>
<td>Outputs the ID of the local player entity</td>
</tr>
</tbody>
</table>

PlayMannequinFragment node

Used to play a Mannequin fragment for the chosen entity with the specified Mannequin tags.

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Play</td>
<td>Any</td>
<td>Plays the fragment</td>
</tr>
<tr>
<td>Fragment</td>
<td>String</td>
<td>Name of the fragment</td>
</tr>
<tr>
<td>Tags</td>
<td>String</td>
<td>List of &quot;+&quot;-separated Mannequin tags</td>
</tr>
<tr>
<td>Priority</td>
<td>Integer</td>
<td>Priority number</td>
</tr>
<tr>
<td>Pause</td>
<td>Any</td>
<td>Pauses the entity actionController</td>
</tr>
<tr>
<td>Resume</td>
<td>Any</td>
<td>Resumes the entity actionController</td>
</tr>
<tr>
<td>ForceFinishLastQueued</td>
<td>Any</td>
<td>Finishes the last queued action</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Trigger for successful fragment command</td>
</tr>
<tr>
<td>Failed</td>
<td>Any</td>
<td>Trigger for failed fragment command</td>
</tr>
</tbody>
</table>
ProcClipEventListener node

Used to listen for a procedural clip event.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Start listening for the procedural clip event</td>
</tr>
<tr>
<td>Stop</td>
<td>Any</td>
<td>Stop listening for the procedural clip event</td>
</tr>
<tr>
<td>Filter</td>
<td>String</td>
<td>Name of the filter used</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>String</td>
<td>Outputs the procedural clip event</td>
</tr>
</tbody>
</table>

AI Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use these flow graph nodes to configure AI agent behaviors and settings.

**Note**
These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at Lumberyard Downloads.

Topics
- ActionAbort node (p. 240)
- ActiveCount node (p. 241)
- ActionEnd node (p. 241)
- ActionStart node (p. 241)
- ActiveCountInFaction node (p. 242)
- ActiveCountMonitor node (p. 242)
- AIGlobalPerceptionScaling node (p. 243)
- AlertMe node (p. 243)
• AttentionTarget node (p. 244)
• AutoDisable node (p. 244)
• Communication node (p. 245)
• EventListener node (p. 245)
• Execute node (p. 246)
• Faction node (p. 247)
• FactionReaction node (p. 247)
• GroupAlertness node (p. 248)
• GroupCount node (p. 249)
• GroupIDGet node (p. 249)
• GroupIDSet node (p. 249)
• IgnoreState node (p. 250)
• IsAliveCheck node (p. 250)
• LookAt node (p. 251)
• NavCostFactor node (p. 251)
• ObjectDrop node (p. 252)
• ObjectGrab node (p. 253)
• ObjectUse node (p. 253)
• PerceptionScale node (p. 254)
• RayCastMNM node (p. 254)
• RegenerateMNM node (p. 255)
• RequestReinforcementReadability node (p. 255)
• SetCommunicationVariable node (p. 256)
• SetFaction node (p. 256)
• SetState node (p. 257)
• ShapeState node (p. 257)
• Signal node (p. 257)
• SmartObjectEvent node (p. 258)
• SmartObjectHelper node (p. 259)
• Stance node (p. 259)

**ActionAbort node**

Used to define a "clean-up" procedure that is run when an AI action is aborted.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abort</td>
<td>Any</td>
<td>Cancels execution of AI action</td>
</tr>
</tbody>
</table>
### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserId</td>
<td>Any</td>
<td>ID of agent that is performing the action</td>
</tr>
<tr>
<td>ObjectId</td>
<td>Any</td>
<td>ID of the object on which the agent is executing</td>
</tr>
</tbody>
</table>

### ActiveCount node

Used to count how many AI agents are active.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>Integer</td>
<td>Number of active agents</td>
</tr>
<tr>
<td>Enemy</td>
<td>Integer</td>
<td>Number of enemies</td>
</tr>
</tbody>
</table>

### ActionEnd node

Used to end an AI action.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>End</td>
<td>Any</td>
<td>Ends the AI action</td>
</tr>
<tr>
<td>Cancel</td>
<td>Any</td>
<td>Cancels the action</td>
</tr>
</tbody>
</table>

### ActionStart node

Used to start an AI action.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserId</td>
<td>Any</td>
<td>ID of agent that is performing the action</td>
</tr>
<tr>
<td>ObjectId</td>
<td>Any</td>
<td>ID of the object on which the agent is executing</td>
</tr>
<tr>
<td>Port</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>ObjectId</td>
<td>Any</td>
<td>ID of the object on which the agent is executing</td>
</tr>
<tr>
<td>Position</td>
<td>Vec3</td>
<td>Position of the object</td>
</tr>
</tbody>
</table>

**ActiveCountInFaction node**

Used to count how many AI factions are active.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faction</td>
<td>String</td>
<td>Fractions to be counted</td>
</tr>
<tr>
<td>IncludedHumanPlayers</td>
<td>Boolean</td>
<td>Include human players when counting active AI agents in the faction</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Integer</td>
<td>Number of active agents in the faction</td>
</tr>
<tr>
<td>Changed</td>
<td>Any</td>
<td>Trigger for changes in the number of active agents</td>
</tr>
</tbody>
</table>

**ActiveCountMonitor node**

Used to monitor the active AI count against a limit and then periodically output the current state. When the condition is met, the monitor loop will stop automatically. This will then need to be restarted manually.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Starts monitoring</td>
</tr>
<tr>
<td>Stops</td>
<td>Any</td>
<td>Stops monitoring</td>
</tr>
<tr>
<td>MaxActiveAIs</td>
<td>Integer</td>
<td>Maximum number of active AIs</td>
</tr>
<tr>
<td>Loop</td>
<td>Boolean</td>
<td>Enables loop monitoring</td>
</tr>
<tr>
<td>Port</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LoopPeriod</td>
<td>Float</td>
<td>Period of time between checks if Loop is enabled</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SlotsFree</td>
<td>Any</td>
<td>Triggers when the number of active agents drops below MaxActiveAIs</td>
</tr>
<tr>
<td>SlotsFull</td>
<td>Any</td>
<td>Triggers when the number of active agents is equal to or above MaxActiveAIs</td>
</tr>
<tr>
<td>CurrentActiveAIs</td>
<td>Integer</td>
<td>Current number of active AI agents</td>
</tr>
</tbody>
</table>

**AIGlobalPerceptionScaling node**

Used to specify a global scale for AI perception.

- Enable
- Disable
- AudioScale
- VisualScale
- FilterAI
- Faction

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables perception scaling</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables perception scaling</td>
</tr>
<tr>
<td>AudioScale</td>
<td>Float</td>
<td>Auditory perception scaling factor</td>
</tr>
<tr>
<td>VisualScale</td>
<td>Float</td>
<td>Visual perception scaling factor</td>
</tr>
<tr>
<td>FilterAI</td>
<td>Integer</td>
<td>Filter which AI agents are used</td>
</tr>
<tr>
<td>Faction</td>
<td>String</td>
<td>Faction</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Any</td>
<td>Triggers when node is enabled</td>
</tr>
<tr>
<td>Disabled</td>
<td>Any</td>
<td>Triggers when node is disabled</td>
</tr>
</tbody>
</table>

**AlertMe node**

A generic AI signal.
### AI Nodes

#### AI:AlertMe

- **Choose Entity**
- **Sync**

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>Any</td>
<td>Generic AI signal</td>
</tr>
</tbody>
</table>

#### AttentionTarget node

Used to output an AI agent's attention target.

- **Choose Entity**
- **Active=1**
- **pos**
- **entityId**
- **none**

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Boolean</td>
<td>Activates the node</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>Vec3</td>
<td>Position of the attention target</td>
</tr>
<tr>
<td>EntityId</td>
<td>Any</td>
<td>Entity ID of attention target</td>
</tr>
<tr>
<td>None</td>
<td>Any</td>
<td>Triggers when there is no attention target</td>
</tr>
</tbody>
</table>

#### AutoDisable node

Used to control auto-disabling.

- **Choose Entity**
- **ON**
- **OFF**

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>On</td>
<td>Any</td>
<td>Enables autodisabling</td>
</tr>
<tr>
<td>Off</td>
<td>Any</td>
<td>Disables autodisabling</td>
</tr>
</tbody>
</table>
Communication node

Used to specify the communication that an AI agent plays.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Starts communication</td>
</tr>
<tr>
<td>Stop</td>
<td>Any</td>
<td>Stops communication</td>
</tr>
<tr>
<td>Communication</td>
<td>String</td>
<td>Name of communication to play</td>
</tr>
<tr>
<td>Channel</td>
<td>String</td>
<td>Name of channel to play the communications in</td>
</tr>
<tr>
<td>ContextExpiry</td>
<td>Float</td>
<td>Time that must elapse before communication can be played again</td>
</tr>
<tr>
<td>SkipSound</td>
<td>Boolean</td>
<td>Skips sound component</td>
</tr>
<tr>
<td>SkipAnim</td>
<td>Boolean</td>
<td>Skips animation component</td>
</tr>
<tr>
<td>TargetId</td>
<td>Any</td>
<td>(Optional) Target ID to play communication at</td>
</tr>
<tr>
<td>TargetPos</td>
<td>Vec3</td>
<td>(Optional) Target position to play communication at</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when communication has finished playing</td>
</tr>
<tr>
<td>Started</td>
<td>Any</td>
<td>Triggers if communication has started</td>
</tr>
<tr>
<td>Stopped</td>
<td>Any</td>
<td>Triggers if communication has stopped</td>
</tr>
<tr>
<td>Finished</td>
<td>Any</td>
<td>Triggers if communication has finished playing</td>
</tr>
<tr>
<td>Fail</td>
<td>Any</td>
<td>Triggers if communication has failed</td>
</tr>
</tbody>
</table>

EventListener node

Used to listen for an event.
### AI: EventListener

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>Vec3</td>
<td>Position of the listener</td>
</tr>
<tr>
<td>Radius</td>
<td>Float</td>
<td>Listening radius of the listener</td>
</tr>
<tr>
<td>ThresholdSound</td>
<td>Float</td>
<td>Sensitivity of the sound output</td>
</tr>
<tr>
<td>ThresholdCollision</td>
<td>Float</td>
<td>Sensitivity of the collision output</td>
</tr>
<tr>
<td>ThresholdBullet</td>
<td>Float</td>
<td>Sensitivity of the bullet output</td>
</tr>
<tr>
<td>ThresholdExplosion</td>
<td>Float</td>
<td>Sensitivity of the explosion output</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sound</td>
<td>Any</td>
<td>Trigger for a sound event</td>
</tr>
<tr>
<td>Collision</td>
<td>Any</td>
<td>Trigger for a collision event</td>
</tr>
<tr>
<td>Bullet</td>
<td>Any</td>
<td>Trigger for a bullet event</td>
</tr>
<tr>
<td>Explosion</td>
<td>Any</td>
<td>Trigger for an explosion event</td>
</tr>
</tbody>
</table>

### Execute node

Used to execute an AI action.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Cancel</td>
<td>Any</td>
<td>Cancels the operation</td>
</tr>
<tr>
<td>Port</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ObjectId</td>
<td>Any</td>
<td>ID of the entity that receives the action that is executed</td>
</tr>
<tr>
<td>Action</td>
<td>String</td>
<td>Action to be executed</td>
</tr>
<tr>
<td>MaxAlertness</td>
<td>Integer</td>
<td>Maximum alertness that allows execution</td>
</tr>
<tr>
<td>HighPriority</td>
<td>Boolean</td>
<td>Action priority level</td>
</tr>
<tr>
<td>Force</td>
<td>Integer</td>
<td>Force execution method</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when the action has been executed</td>
</tr>
<tr>
<td>Succeed</td>
<td>Any</td>
<td>Triggers if the action has been executed</td>
</tr>
<tr>
<td>Fail</td>
<td>Any</td>
<td>Triggers if the action has not been executed</td>
</tr>
</tbody>
</table>

### Faction node

Used to trigger an AI faction.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Triggers the output</td>
</tr>
<tr>
<td>Faction</td>
<td>String</td>
<td>Name of faction to trigger</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faction</td>
<td>String</td>
<td>Outputs the faction that was triggered</td>
</tr>
</tbody>
</table>

### FactionReaction node

Used to set or get AI faction reaction information.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>Neutral</td>
<td>Reaction of the source</td>
</tr>
<tr>
<td>Target</td>
<td>Friendly</td>
<td>Reaction of the target</td>
</tr>
<tr>
<td>Reaction=Hostile</td>
<td>Hostile</td>
<td>Reaction of the target</td>
</tr>
<tr>
<td>Get</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Lumberyard Legacy Reference

### AI Nodes

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>String</td>
<td>Source faction</td>
</tr>
<tr>
<td>Target</td>
<td>String</td>
<td>Target faction</td>
</tr>
<tr>
<td>Reaction</td>
<td>Integer</td>
<td>Source faction reaction to target faction</td>
</tr>
<tr>
<td>Get</td>
<td>Any</td>
<td>Gets the faction reaction and triggers output</td>
</tr>
<tr>
<td>Set</td>
<td>Any</td>
<td>Sets the faction reaction and triggers output</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neutral</td>
<td>Boolean</td>
<td>Triggers if source faction reaction to target faction is neutral</td>
</tr>
<tr>
<td>Friendly</td>
<td>Boolean</td>
<td>Triggers if source faction reaction to target faction is friendly</td>
</tr>
<tr>
<td>Hostile</td>
<td>Boolean</td>
<td>Triggers if source faction reaction to target faction is hostile</td>
</tr>
</tbody>
</table>

### GroupAlertness node

Used to output the alertness level of any AI agent in a group.

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupId</td>
<td>Integer</td>
<td>ID of group to set alertness level for</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alertness</td>
<td>Integer</td>
<td>Alertness level of the group</td>
</tr>
<tr>
<td>Green</td>
<td>Any</td>
<td>Trigger for green alertness level</td>
</tr>
<tr>
<td>Orange</td>
<td>Any</td>
<td>Trigger for orange alertness level</td>
</tr>
<tr>
<td>Red</td>
<td>Any</td>
<td>Trigger for red alertness level</td>
</tr>
<tr>
<td>PlayerSighted</td>
<td>Any</td>
<td>Triggers if the player has been sighted</td>
</tr>
</tbody>
</table>

248
**GroupCount node**

Used to output the AI agent count in a group.

![GroupCount node](image)

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupId</td>
<td>Integer</td>
<td>Agent group ID</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Integer</td>
<td>Number of agents in the group</td>
</tr>
<tr>
<td>Empty</td>
<td>Any</td>
<td>Triggers if no agents are in the group</td>
</tr>
</tbody>
</table>

**GroupIDGet node**

Used to output the group ID for an AI agent.

![GroupIDGet node](image)

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>Any</td>
<td>Triggers the output</td>
</tr>
<tr>
<td>GroupId</td>
<td>Integer</td>
<td>Group ID</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GroupId</td>
<td>Integer</td>
<td>Outputs agent group ID</td>
</tr>
</tbody>
</table>
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>Any</td>
<td>Triggers the output</td>
</tr>
<tr>
<td>FromId</td>
<td>Integer</td>
<td>The group to be merged</td>
</tr>
<tr>
<td>Told</td>
<td>Integer</td>
<td>The group to merge to</td>
</tr>
<tr>
<td>EnabledFromGroup</td>
<td>Boolean</td>
<td>Enables members of the FromId group</td>
</tr>
</tbody>
</table>

**IgnoreState node**

Used to make an AI agent ignore enemies or to be ignored.

![IgnoreState node diagram]

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hostile</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Ignore</td>
<td>Any</td>
<td>Agent ignores enemies</td>
</tr>
<tr>
<td>ResetPerception</td>
<td>Any</td>
<td>Resets Ignore state</td>
</tr>
</tbody>
</table>

**IsAliveCheck node**

Used to check which AI actors of a group are active.

![IsAliveCheck node diagram]

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>AliveCount</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>Actor 0</td>
<td>AliveId 0</td>
<td>Specific actors to check</td>
</tr>
<tr>
<td>Actor 1</td>
<td>AliveId 1</td>
<td>Specific actors to check</td>
</tr>
<tr>
<td>Actor 2</td>
<td>AliveId 2</td>
<td>Specific actors to check</td>
</tr>
<tr>
<td>Actor 3</td>
<td>AliveId 3</td>
<td>Specific actors to check</td>
</tr>
<tr>
<td>Actor 4</td>
<td>AliveId 4</td>
<td>Specific actors to check</td>
</tr>
<tr>
<td>Actor 5</td>
<td>AliveId 5</td>
<td>Specific actors to check</td>
</tr>
<tr>
<td>Actor 6</td>
<td>AliveId 6</td>
<td>Specific actors to check</td>
</tr>
<tr>
<td>Actor 7</td>
<td>AliveId 7</td>
<td>Specific actors to check</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AliveCount</td>
<td>Integer</td>
<td>Number of actors that are alive</td>
</tr>
<tr>
<td>AliveId0 - 7</td>
<td>Any</td>
<td>Triggers if specific actor is alive</td>
</tr>
</tbody>
</table>

LookAt node

Used to make an AI agent look at a specific location, an entity, or a direction.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Cancel</td>
<td>Any</td>
<td>Cancels the operation</td>
</tr>
<tr>
<td>Point</td>
<td>Vec3</td>
<td>Point for agent to look at</td>
</tr>
<tr>
<td>Direction</td>
<td>Vec3</td>
<td>Direction for agent to look along</td>
</tr>
<tr>
<td>ObjectId</td>
<td>Any</td>
<td>ID of object for agent to look at</td>
</tr>
<tr>
<td>Duration</td>
<td>Float</td>
<td>Time in seconds for agent to look</td>
</tr>
<tr>
<td>Force</td>
<td>Integer</td>
<td>Force execution method</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when agent is done looking</td>
</tr>
<tr>
<td>Succeed</td>
<td>Any</td>
<td>Triggers if agent is looking</td>
</tr>
<tr>
<td>Fail</td>
<td>Any</td>
<td>Triggers if agent is not looking</td>
</tr>
</tbody>
</table>

NavCostFactor node

Used to set the AI navigation cost factor for traveling through a region.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Factor</td>
<td>Float</td>
<td>Navigation cost factor</td>
</tr>
<tr>
<td>NavModifierName</td>
<td>String</td>
<td>Name of the cost factor navigation modifier</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Integer</td>
<td>Triggers when cost factor for travelling through a region has been set</td>
</tr>
</tbody>
</table>

ObjectDrop node

Used to have an AI agent drop a grabbed object.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Cancel</td>
<td>Any</td>
<td>Cancels the operation</td>
</tr>
<tr>
<td>Impulse</td>
<td>Vec3</td>
<td>Impulse strength for dropping object</td>
</tr>
<tr>
<td>Force</td>
<td>Integer</td>
<td>Force execution method</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when object has been dropped</td>
</tr>
<tr>
<td>Succeed</td>
<td>Any</td>
<td>Triggers if object is dropped</td>
</tr>
<tr>
<td>Fail</td>
<td>Any</td>
<td>Triggers if object is not dropped</td>
</tr>
</tbody>
</table>
ObjectGrab node

Used to make an AI agent grab an object.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Cancel</td>
<td>Any</td>
<td>Cancels the operation</td>
</tr>
<tr>
<td>ObjectId</td>
<td>Any</td>
<td>Object to be grabbed</td>
</tr>
<tr>
<td>Force</td>
<td>Integer</td>
<td>Force execution method</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when object has been grabbed</td>
</tr>
<tr>
<td>Succeed</td>
<td>Any</td>
<td>Triggers if object is grabbed</td>
</tr>
<tr>
<td>Fail</td>
<td>Any</td>
<td>Triggers if object is not grabbed</td>
</tr>
</tbody>
</table>

ObjectUse node

Used to make an AI agent use an object.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Cancel</td>
<td>Any</td>
<td>Cancels the operation</td>
</tr>
<tr>
<td>ObjectId</td>
<td>Any</td>
<td>Object to be used</td>
</tr>
<tr>
<td>Force</td>
<td>Integer</td>
<td>Force execution method</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Cancel</td>
<td>Any</td>
<td>Cancels the operation</td>
</tr>
<tr>
<td>ObjectId</td>
<td>Any</td>
<td>Object to be used</td>
</tr>
<tr>
<td>Force</td>
<td>Integer</td>
<td>Force execution method</td>
</tr>
</tbody>
</table>
### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when object has been used</td>
</tr>
<tr>
<td>Succeed</td>
<td>Any</td>
<td>Triggers if object is used</td>
</tr>
<tr>
<td>Fail</td>
<td>Any</td>
<td>Triggers if object is not used</td>
</tr>
</tbody>
</table>

### PerceptionScale node

Used to scale the perception for an AI agent.

![AI:PerceptionScale](image)

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>Visual</td>
<td>Float</td>
<td>Visual perception scale factor</td>
</tr>
<tr>
<td>Audio</td>
<td>Float</td>
<td>Auditory perception scale factor</td>
</tr>
</tbody>
</table>

### RayCastMNM node

Performs a raycast to the AI multilayer navigation mesh relative to an entity.

![AI:RayCastMNM](image)

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choose Entity</td>
<td>Any</td>
<td>Changes the attached entity dynamically</td>
</tr>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>Direction</td>
<td>Vec3</td>
<td>The direction of the raycast</td>
</tr>
<tr>
<td>MaxLength</td>
<td>Float</td>
<td>The maximum length of the raycast</td>
</tr>
</tbody>
</table>
Lumberyard Legacy Reference
AI Nodes

### Port

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>Vec3</td>
<td>The ray start position, relative to the entity</td>
</tr>
<tr>
<td>TransformDirection</td>
<td>Boolean</td>
<td>Whether the direction is transformed by entity orientation</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoHit</td>
<td>Any</td>
<td>Activated if no object was hit by the raycast</td>
</tr>
<tr>
<td>Hit</td>
<td>Any</td>
<td>Activated if an object was hit by the raycast</td>
</tr>
<tr>
<td>RayDirection</td>
<td>Vector</td>
<td>Actual direction of the cast ray, possibly transformed by entity rotation (assumes Hit)</td>
</tr>
<tr>
<td>HitDistance</td>
<td>Float</td>
<td>Distance to the hit object (assumes Hit)</td>
</tr>
<tr>
<td>HitPoint</td>
<td>Vector</td>
<td>Position of the hit (assumes Hit)</td>
</tr>
<tr>
<td>MeshId</td>
<td>Integer</td>
<td>The mesh ID of the navigation mesh hit</td>
</tr>
</tbody>
</table>

### RegenerateMNM node

Used to regenerate the AI multi-navigation mesh.

![AI:RegenerateMNM]

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Triggers recalculation of MNM data for the bounding box</td>
</tr>
<tr>
<td>Min</td>
<td>Vec3</td>
<td>Minimum limit of bounding box</td>
</tr>
<tr>
<td>Max</td>
<td>Vec3</td>
<td>Maximum limit of bounding box</td>
</tr>
</tbody>
</table>

### RequestReinforcementReadability node

Used to make an AI agent request reinforcements. There is no guarantee that the action will be performed however.

![AI:RequestReinforcementReadability]

255
## AI Nodes

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>GroupId</td>
<td>Integer</td>
<td>ID of the group that is notified</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when group has been notified</td>
</tr>
</tbody>
</table>

### SetCommunicationVariable node

Used to set the communication variable that an AI agent uses to communicate their intentions.

```plaintext
AI:SetCommunicationVariable
  ➤ Set
  ➤ VariableName= \( \text{VariableName} \)
  ➤ VariableValue=1
```

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>Any</td>
<td>Sets the variable</td>
</tr>
<tr>
<td>VariableName</td>
<td>String</td>
<td>Variable to be set</td>
</tr>
<tr>
<td>VariableValue</td>
<td>Boolean</td>
<td>Value of variable</td>
</tr>
</tbody>
</table>

### SetFaction node

Used to set the faction that an AI agent belongs to.

```plaintext
AI:SetFaction
  ➤ Choose Entity
  ➤ Faction= \( \text{Faction} \)
  ➤ SetToDefault=0
  ➤ Set
```

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faction</td>
<td>String</td>
<td>Faction to be set</td>
</tr>
<tr>
<td>SetToDefault</td>
<td>Boolean</td>
<td>Set to default faction</td>
</tr>
<tr>
<td>Set</td>
<td>Any</td>
<td>Sets the faction for the agent</td>
</tr>
</tbody>
</table>
**SetState node**

Used to set the Smart Object state for an AI agent.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>State</td>
<td>String</td>
<td>Smart object state to be set</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when state has been set</td>
</tr>
<tr>
<td>Succeed</td>
<td>Any</td>
<td>Triggers if state has been set</td>
</tr>
<tr>
<td>Fail</td>
<td>Any</td>
<td>Triggers if state has not been set</td>
</tr>
</tbody>
</table>

**ShapeState node**

Use to enable or disable an AI shape.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the AI shape</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the AI shape</td>
</tr>
<tr>
<td>ShapeName</td>
<td>String</td>
<td>Name of the AI shape</td>
</tr>
</tbody>
</table>
AI Signal node

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Signal</td>
<td>String</td>
<td>Name of the signal to be sent</td>
</tr>
<tr>
<td>PosValue</td>
<td>Vec3</td>
<td>Position value 1 of the signal</td>
</tr>
<tr>
<td>PosValue2</td>
<td>Vec3</td>
<td>Position value 2 of the signal</td>
</tr>
<tr>
<td>IValue</td>
<td>Integer</td>
<td>Integer value of the signal</td>
</tr>
<tr>
<td>FValue</td>
<td>Float</td>
<td>Floating point value of the signal</td>
</tr>
<tr>
<td>SValue</td>
<td>String</td>
<td>String value of the signal</td>
</tr>
<tr>
<td>Id</td>
<td>Any</td>
<td>ID of the signal</td>
</tr>
<tr>
<td>Force</td>
<td>Boolean</td>
<td>Force execution method</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when the signal has been sent</td>
</tr>
<tr>
<td>Succeed</td>
<td>Any</td>
<td>Triggers if the signal is sent</td>
</tr>
<tr>
<td>Fail</td>
<td>Any</td>
<td>Triggers if the signal is not sent</td>
</tr>
</tbody>
</table>

SmartObjectEvent node

Used to trigger a smart object event.

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>userId</td>
<td></td>
</tr>
<tr>
<td>trigger</td>
<td>objectId</td>
<td></td>
</tr>
<tr>
<td>userId</td>
<td>start</td>
<td></td>
</tr>
<tr>
<td>objectId</td>
<td>noRule</td>
<td></td>
</tr>
</tbody>
</table>
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>String</td>
<td>Smart object event to be triggered</td>
</tr>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Triggers the event</td>
</tr>
<tr>
<td>UserId</td>
<td>Any</td>
<td>Limits event to specific user ID</td>
</tr>
<tr>
<td>ObjectId</td>
<td>Any</td>
<td>Limits event to specific object</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserId</td>
<td>Any</td>
<td>ID of the user that receives the event</td>
</tr>
<tr>
<td>ObjectId</td>
<td>Any</td>
<td>ID of the object that receives the event</td>
</tr>
<tr>
<td>Start</td>
<td>Any</td>
<td>Triggers if matching rule is found</td>
</tr>
<tr>
<td>NoRule</td>
<td>Any</td>
<td>Triggers if no matching rule is found</td>
</tr>
</tbody>
</table>

SmartObjectHelper node

Used to output an AI agent's attention target parameter.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>String</td>
<td>Class of smart object helper</td>
</tr>
<tr>
<td>Helper</td>
<td>String</td>
<td>Name of smart object helper</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>Vec3</td>
<td>Position of smart object helper</td>
</tr>
<tr>
<td>Fwd</td>
<td>Vec3</td>
<td>Forward direction of smart object helper</td>
</tr>
<tr>
<td>Up</td>
<td>Vec3</td>
<td>Up direction of smart object helper</td>
</tr>
</tbody>
</table>

Stance node

Used to control an AI agent's body stance.
You can use these flow graph nodes to configure AI sequence behaviors and settings. All AI sequence nodes must be executed with the **AISequence:Start** node.

**Note**
These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at Lumberyard Downloads.

**Topics**
- Animation node (p. 261)
- ApproachAndEnterVehicle node (p. 262)
- Bookmark node (p. 262)
- End node (p. 263)
- HoldFormation node (p. 263)
- JoinFormation node (p. 263)
- Move node (p. 264)
Animation node

Used to make an AI agent move to a specific location and play an animation.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Starts the animation</td>
</tr>
<tr>
<td>Stop</td>
<td>Any</td>
<td>Stops the animation</td>
</tr>
<tr>
<td>Animation</td>
<td>String</td>
<td>Name of the animation</td>
</tr>
<tr>
<td>DestinationEntity</td>
<td>Any</td>
<td>Destination to move to</td>
</tr>
<tr>
<td>Position</td>
<td>Vec3</td>
<td>Position to move to</td>
</tr>
<tr>
<td>Direction</td>
<td>Vec3</td>
<td>Direction to move along</td>
</tr>
<tr>
<td>Speed</td>
<td>Integer</td>
<td>Speed of movement</td>
</tr>
<tr>
<td>Stance</td>
<td>Integer</td>
<td>Stance of the agent while moving</td>
</tr>
<tr>
<td>OneShot</td>
<td>Boolean</td>
<td>True for a one-shot animation, false for a looping animation</td>
</tr>
<tr>
<td>StartRadius</td>
<td>Float</td>
<td>Start radius</td>
</tr>
<tr>
<td>DirectionTolerance</td>
<td>Float</td>
<td>Direction tolerance</td>
</tr>
<tr>
<td>LoopDuration</td>
<td>Float</td>
<td>Duration of the looping animation; ignored for a one-shot animation</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Outputs when the animation has completed</td>
</tr>
</tbody>
</table>
ApproachAndEnterVehicle node

Used to make an AI agent approach and then enter a vehicle.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Allows the AI agent to move to, and enter the specified vehicle</td>
</tr>
<tr>
<td>VehicleID</td>
<td>Any</td>
<td>Vehicle to be entered</td>
</tr>
<tr>
<td>Seat</td>
<td>Integer</td>
<td>Seat to be entered</td>
</tr>
<tr>
<td>Speed</td>
<td>Integer</td>
<td>Speed at which the AI agent approaches the vehicle</td>
</tr>
<tr>
<td>Stance</td>
<td>Integer</td>
<td>Stance of the agent when approaching the vehicle</td>
</tr>
<tr>
<td>Fast</td>
<td>Boolean</td>
<td>Skips the approaching animation</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Output when the AI agent has completed entering the vehicle</td>
</tr>
</tbody>
</table>

Bookmark node

Used to define a bookmark in a sequence of AI actions from which the sequence will resume after being interrupted.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>Any</td>
<td>Sets a bookmark for the AI sequence from which to resume from</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link</td>
<td>Any</td>
<td>Link to other nodes</td>
</tr>
</tbody>
</table>
End node

Used to define the end of a sequence of AI actions. This frees the AI agent to resume typical behaviors.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>End</td>
<td>Any</td>
<td>Triggers when the AI sequence ends</td>
</tr>
</tbody>
</table>

HoldFormation node

Use to create a formation to have an AI agent hold to.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Start formation hold</td>
</tr>
<tr>
<td>FormationName</td>
<td>String</td>
<td>Name of the formation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when the formation is complete</td>
</tr>
</tbody>
</table>

JoinFormation node

Use to have an AI agent join a formation.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Start formation join</td>
</tr>
<tr>
<td>LeaderId</td>
<td>Any</td>
<td>ID of the leader</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when the formation is complete</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when formation join is complete</td>
</tr>
</tbody>
</table>

**Move node**

Use to command an AI agent to move to a location.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Start movement</td>
</tr>
<tr>
<td>Speed</td>
<td>Integer</td>
<td>Movement speed</td>
</tr>
<tr>
<td>Stance</td>
<td>Integer</td>
<td>Stance of the agent while moving</td>
</tr>
<tr>
<td>DestinationEntity</td>
<td>Any</td>
<td>Destination entity to move to</td>
</tr>
<tr>
<td>Position</td>
<td>Vec3</td>
<td>Position to move to</td>
</tr>
<tr>
<td>Direction</td>
<td>Vec3</td>
<td>Direction to move along</td>
</tr>
<tr>
<td>EndDistance</td>
<td>Float</td>
<td>End distance to move to</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when movement is complete</td>
</tr>
</tbody>
</table>

**MoveAlongPath node**

Use to have an AI agent move along a path.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Begins AI agent movement</td>
</tr>
<tr>
<td>Port</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Speed</td>
<td>Integer</td>
<td>Speed of the agent</td>
</tr>
<tr>
<td>Stance</td>
<td>Integer</td>
<td>Stance of the agent while following the path</td>
</tr>
<tr>
<td>PathName</td>
<td>String</td>
<td>Name of the path the agent follows</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when the agent has completed the path</td>
</tr>
</tbody>
</table>

**Shoot node**

Use to make an AI agent shoot at an entity or a location for a specified length of time.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Starts the AI agent shooting</td>
</tr>
<tr>
<td>TargetEntity</td>
<td>Any</td>
<td>Entity the agent shoots at</td>
</tr>
<tr>
<td>TargetPosition</td>
<td>Vec3</td>
<td>Position the agent shoots at</td>
</tr>
<tr>
<td>Duration</td>
<td>Float</td>
<td>Length of shooting time</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when shooting is finished</td>
</tr>
</tbody>
</table>

**Stance node**

Use to control the stance of an AI agent.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Starts the stance</td>
</tr>
<tr>
<td>Port</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>Stance</td>
<td>Integer</td>
<td>Name of the AI agent stance</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when the stance is complete</td>
</tr>
</tbody>
</table>

**Start node**

Use to define the start of an AI sequence of actions. All AI sequence nodes must be executed using this node.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Starts the AI sequence</td>
</tr>
<tr>
<td>Interruptible</td>
<td>Boolean</td>
<td>Automatically stops the agent when not in the Idle state.</td>
</tr>
<tr>
<td>ResumeAfterInterruption</td>
<td>Boolean</td>
<td>When the Idle state ends, AI sequence automatically resumes from the start or from the agent's bookmark ID</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>TypedList</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link</td>
<td>Any</td>
<td>Link to other nodes</td>
</tr>
</tbody>
</table>

**VehicleRotateTurret node**

Use to rotate a vehicle turret to an aiming position.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Starts turret rotation</td>
</tr>
<tr>
<td>AimPos</td>
<td>Vec3</td>
<td>Position that the turret aims a</td>
</tr>
</tbody>
</table>
Lumberyard Legacy Reference
Animations Nodes

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ThresholdPitch</td>
<td>Float</td>
<td>Pitch angle threshold at which the output port is triggered; must be used with the ThresholdYaw port</td>
</tr>
<tr>
<td>ThresholdYaw</td>
<td>Float</td>
<td>Yaw angle threshold at which the output port is triggered; must be used with the ThresholdPitch port</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Starts turret rotation</td>
</tr>
</tbody>
</table>

**Wait node**

Used to make the AI agent wait for a specified length of time.

**Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.**

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use these flow graph nodes to configure animation-related settings.

**Topics**

267
• AnimationEventListener node (p. 268)
• AttachmentControl node (p. 268)
• BonelInfo node (p. 269)
• CheckAnimPlaying node (p. 269)
• CooperativeAnimation node (p. 270)
• LookAt node (p. 271)
• NoAiming node (p. 272)
• PlayAnimation node (p. 272)
• PlayCGA node (p. 273)
• PlaySequence node (p. 274)
• StopAnimation node (p. 275)
• SynchronizeTwoAnimations node (p. 275)
• TriggerOnKeyTime node (p. 276)

**AnimationEventListener node**

Use to listen for a specific animation event and trigger the output.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Starts listening for animation events</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Stops listening for animation events</td>
</tr>
<tr>
<td>EventName</td>
<td>String</td>
<td>Name of the animation event to listen for</td>
</tr>
<tr>
<td>Once</td>
<td>Boolean</td>
<td>If set to 1 (true), the node is disabled after the event has been received.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Any</td>
<td>Triggers when listening has started</td>
</tr>
<tr>
<td>Disabled</td>
<td>Any</td>
<td>Triggers when listening has stopped</td>
</tr>
<tr>
<td>EventTriggered</td>
<td>Any</td>
<td>Triggers when the animation event is received</td>
</tr>
</tbody>
</table>

**AttachmentControl node**

Use to add and control an attachment for a character.
Lumberyard Legacy Reference
Animations Nodes

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attachment</td>
<td>String</td>
<td>Name of the attachment</td>
</tr>
<tr>
<td>Show</td>
<td>Any</td>
<td>Shows the attachment</td>
</tr>
<tr>
<td>Hide</td>
<td>Any</td>
<td>Hides the attachment</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shown</td>
<td>Any</td>
<td>Triggers when the attachment is shown</td>
</tr>
<tr>
<td>Hidden</td>
<td>Any</td>
<td>Triggers when the attachment is hidden</td>
</tr>
</tbody>
</table>

BoneInfo node

Use to specify and output character bones to create attachments or link objects.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BoneName</td>
<td>String</td>
<td>Name of the bone to get information for</td>
</tr>
<tr>
<td>Enabled</td>
<td>Boolean</td>
<td>Enables and disables the node</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LocalPos</td>
<td>Vec3</td>
<td>Position of the bone in local space</td>
</tr>
<tr>
<td>LocalRot</td>
<td>Vec3</td>
<td>Rotation of the bone in local space</td>
</tr>
<tr>
<td>WorldPos</td>
<td>Vec3</td>
<td>Position of the bone in world space</td>
</tr>
<tr>
<td>WorldRot</td>
<td>Vec3</td>
<td>Rotation of the bone in world space</td>
</tr>
</tbody>
</table>

CheckAnimPlaying node

Use to check whether a defined animation is playing or not.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check</td>
<td>Any</td>
<td>Checks once whether the animation is playing</td>
</tr>
<tr>
<td>CheckAlways</td>
<td>Boolean</td>
<td>Checks each frame whether the animation is playing</td>
</tr>
<tr>
<td>Animation</td>
<td>String</td>
<td>Name of the animation</td>
</tr>
<tr>
<td>Layer</td>
<td>Integer</td>
<td>Specifies which layer should play the animation</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Playing</td>
<td>Any</td>
<td>Triggers when the animation is playing on the layer</td>
</tr>
<tr>
<td>NotPlaying</td>
<td>Any</td>
<td>Triggers when the animation is not playing on the layer</td>
</tr>
<tr>
<td>TopOfStack</td>
<td>Any</td>
<td>Triggers when the animation is at the top of the stack, meaning it is not currently blended out</td>
</tr>
</tbody>
</table>

**CooperativeAnimation node**

Use to allow the playing of a positioned and aligned animation for one or more characters.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Starts the animation</td>
</tr>
<tr>
<td>Stop</td>
<td>Any</td>
<td>Stops the animation</td>
</tr>
</tbody>
</table>
## Animations Nodes

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ForceStart</td>
<td></td>
<td>Force the animation to start</td>
</tr>
<tr>
<td>AdjustToTerrain</td>
<td></td>
<td>Makes sure the character is at terrain level</td>
</tr>
<tr>
<td>IgnoreCharactersDeath</td>
<td></td>
<td>If false and any of the actors die, stops the animation for all the actors</td>
</tr>
<tr>
<td>NoCollisionBetween</td>
<td></td>
<td>If true, the first actor won't collide with the other actors</td>
</tr>
<tr>
<td>Location</td>
<td></td>
<td>Starts the animation at a specific location</td>
</tr>
<tr>
<td>Rotation</td>
<td></td>
<td>Starts the animation at a specific rotation</td>
</tr>
<tr>
<td>Alignment</td>
<td></td>
<td>Alignment type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>WildMatch</strong>: Moves both characters the least amount</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>FirstActor</strong>: Allows first actor to be rotated but not moved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>FirstActorNoRot</strong>: Prevents first actor from being moved or rotated</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>FirstActorPosition</strong>: Slides the actor so that the first one is at the specified Location</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Location</strong>: Moves both characters until the reference point of the animation is at the specified location</td>
</tr>
<tr>
<td>Entity_01 - Entity_04</td>
<td></td>
<td>Name of the specific entity</td>
</tr>
<tr>
<td>AnimationName_01 - AnimationName_4</td>
<td></td>
<td>Name of the specific animation</td>
</tr>
<tr>
<td>SlideDuration_01 - SlideDuration_04</td>
<td></td>
<td>Time in seconds to slide the entity into position</td>
</tr>
<tr>
<td>HPhysics1 - HPhysics4</td>
<td></td>
<td>Prohibits the character from being pushed through solid objects</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finished_01 - Finished_04</td>
<td>Any</td>
<td>Triggers when the specific actor is done</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when all actors are done</td>
</tr>
</tbody>
</table>

### LookAt node

Use to make a character look at a position.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Character begins looking at a target</td>
</tr>
<tr>
<td>Stop</td>
<td>Any</td>
<td>Character stops looking at a target</td>
</tr>
<tr>
<td>FieldOfView</td>
<td>Float</td>
<td>Field of view for the character</td>
</tr>
<tr>
<td>Blending</td>
<td>Float</td>
<td></td>
</tr>
<tr>
<td>Target</td>
<td>Any</td>
<td>Target for the character to look at</td>
</tr>
<tr>
<td>TargetPos</td>
<td>Vec3</td>
<td>Target look position</td>
</tr>
<tr>
<td>LookAtPlayer</td>
<td>Boolean</td>
<td>Character looks at player</td>
</tr>
</tbody>
</table>

NoAiming node

Use to suppress aiming for a character.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dont Aim!</td>
<td>Any</td>
<td>Suppresses aiming for a character</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td></td>
</tr>
</tbody>
</table>

PlayAnimation node

Use to play an animation on the character’s skeleton, bypassing the AnimationGraph. The animation name can be specified directly as mapped in the .cal file.
## Animations Nodes

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Starts the animation</td>
</tr>
<tr>
<td>Stop</td>
<td>Any</td>
<td>Stops the animation</td>
</tr>
<tr>
<td>Animation</td>
<td>String</td>
<td>Name of the animation to play</td>
</tr>
<tr>
<td>BlendInTime</td>
<td>Float</td>
<td>Blend-in time in seconds</td>
</tr>
<tr>
<td>Layer</td>
<td>Integer</td>
<td>Layer on which to play the animation</td>
</tr>
<tr>
<td>Loop</td>
<td>Boolean</td>
<td>Loops the animation indefinitely</td>
</tr>
<tr>
<td>ForceUpdate</td>
<td>Boolean</td>
<td>Plays animation even if not visible</td>
</tr>
<tr>
<td>PauseAnimGraph</td>
<td>Boolean</td>
<td>Deprecated</td>
</tr>
<tr>
<td>ControlMovement</td>
<td>Boolean</td>
<td>Controls movement of the entities</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when the animation is done</td>
</tr>
<tr>
<td>AlmostDone</td>
<td>Any</td>
<td>Triggers when the animation is almost done</td>
</tr>
</tbody>
</table>

### PlayCGA node

Use to play .cga files and their animation, as well as .anm files belonging to the .cga file. The Trigger input starts the animation.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CGA_File</td>
<td>String</td>
<td>File name of the animation</td>
</tr>
<tr>
<td>CGA_Animation</td>
<td>String</td>
<td>Name of the animation</td>
</tr>
<tr>
<td>Trigger</td>
<td>Boolean</td>
<td>Starts the animation</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Triggers when the animation has finished</td>
</tr>
</tbody>
</table>
# PlaySequence node

Use to play a track view sequence. Use the PerformBlendOut input to make sure that the camera has a seamless blend into the game camera when the sequence is over. Make sure to beam the player to the right place when the sequence starts.

## Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence</td>
<td>String</td>
<td>Name of the sequence</td>
</tr>
<tr>
<td>StartTrigger</td>
<td>Any</td>
<td>Starts the sequence</td>
</tr>
<tr>
<td>PauseTrigger</td>
<td>Any</td>
<td>Pauses the sequence</td>
</tr>
<tr>
<td>StopTrigger</td>
<td>Any</td>
<td>Stops the sequence</td>
</tr>
<tr>
<td>PrecacheTrigger</td>
<td>Any</td>
<td>Precaches keys that start in the first few seconds of the animation</td>
</tr>
<tr>
<td>BreakOnStop</td>
<td>Boolean</td>
<td>If set to true, stopping the sequence doesn’t jump it to the end</td>
</tr>
<tr>
<td>BlendPosSpeed</td>
<td>Float</td>
<td>Speed at which the position gets blended into the animation</td>
</tr>
<tr>
<td>BlendRotSpeed</td>
<td>Float</td>
<td>Speed at which the rotation gets blended into the animation</td>
</tr>
<tr>
<td>PerformBlendOut</td>
<td>Boolean</td>
<td>If true, the end of the cut scene dissolves to the new view</td>
</tr>
<tr>
<td>StartTime</td>
<td>Float</td>
<td>Time at which the sequence starts playing</td>
</tr>
<tr>
<td>PlaySpeed</td>
<td>Float</td>
<td>Speed at which the sequence plays</td>
</tr>
<tr>
<td>JumpToTime</td>
<td>Float</td>
<td>Jumps to a specific time in the sequence</td>
</tr>
<tr>
<td>TriggerJumpToTime</td>
<td>Any</td>
<td>Triggers the animation to jump to the specified sequence time</td>
</tr>
<tr>
<td>TriggerJumpToEnd</td>
<td>Any</td>
<td>Triggers the animation to jump to the end of the sequence</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Started</td>
<td>Any</td>
<td>Triggers when the animation starts</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when the animation has stopped or is aborted</td>
</tr>
<tr>
<td>Finished</td>
<td>Any</td>
<td>Triggers when the animation has stopped</td>
</tr>
<tr>
<td>Aborted</td>
<td>Any</td>
<td>Triggers when the animation is aborted</td>
</tr>
<tr>
<td>SequenceTime</td>
<td>Float</td>
<td>Current time of the sequence</td>
</tr>
<tr>
<td>CurrentSpeed</td>
<td>Float</td>
<td>Speed at which the sequence is being played</td>
</tr>
</tbody>
</table>

StopAnimation node

Use to stop the animation.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop!</td>
<td>Any</td>
<td>Stops the animation</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when the animation has stopped</td>
</tr>
</tbody>
</table>

SynchronizeTwoAnimations node

Use to synchronize two animations for two entities.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity1</td>
<td>Any</td>
<td>First entity to synchronize</td>
</tr>
<tr>
<td>Entity2</td>
<td>Any</td>
<td>Second entity to synchronize</td>
</tr>
<tr>
<td>Animation1</td>
<td>String</td>
<td>First animation to synchronize</td>
</tr>
<tr>
<td>Animation2</td>
<td>String</td>
<td>Second animation to synchronize</td>
</tr>
<tr>
<td>Port</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>ResyncTime</td>
<td>Float</td>
<td>Resync time</td>
</tr>
<tr>
<td>MaxPercentSpeedChange</td>
<td>Float</td>
<td>Maximum percentage speed change</td>
</tr>
</tbody>
</table>

**TriggerOnKeyTime node**

Use to play and output an animation at a specified time.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Animation</td>
<td>String</td>
<td>Animation to play</td>
</tr>
<tr>
<td>TriggerTime</td>
<td>Float</td>
<td>When to play the animation</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Plays the animation</td>
</tr>
</tbody>
</table>

**Audio Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

The following flow graph nodes are used to control various audio system functionality and settings. All references to Audiokinetic Wwise also applies to the LTX version.

Topics

- entity:AudioTriggerSpot node (p. 277)
- entity:AudioAreaEntity node (p. 277)
- entity:AudioAreaAmbience node (p. 278)
- entity:AudioAreaRandom node (p. 278)
- PreloadData node (p. 278)
- Rtpc node (p. 279)
- Switch node (p. 279)
- Trigger node (p. 280)
entity: AudioTriggerSpot node

Used to enable and disable the associated entity.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Stops the sound. If available, triggers the event set in the StopTriggerName property.</td>
</tr>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Starts the sound. Triggers the event set in the PlayTriggerName property.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Triggers when the event started by the play trigger has completed playing.</td>
</tr>
</tbody>
</table>

entity: AudioAreaEntity node

Used to enable and disable the associated entity, as well as control what happens when the player enters and leaves the shape.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Stops the sound. If available, triggers the event set in the StopTriggerName property.</td>
</tr>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Starts the sound. Triggers the event set in the PlayTriggerName property.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FadeValue</td>
<td>Float</td>
<td>Normalized value from 0 to 1 of the FadeDistance when the player approaches the shape</td>
</tr>
<tr>
<td>OnFarToNear</td>
<td>Boolean</td>
<td>Triggers when player enters the fade distance</td>
</tr>
<tr>
<td>OnInsideToNear</td>
<td>Boolean</td>
<td>Triggers when player leaves the shape</td>
</tr>
<tr>
<td>Port</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>OnNearToFar</td>
<td>Boolean</td>
<td>Triggers when player leaves the fade distance</td>
</tr>
<tr>
<td>OnNearToInside</td>
<td>Boolean</td>
<td>Triggers when player enters the shape</td>
</tr>
</tbody>
</table>

**entity:AudioAreaAmbience node**

Used to enable and disable the associated entity.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Disables the audio entity</td>
</tr>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Enables the audio entity</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Triggers when the audio entity is enabled</td>
</tr>
</tbody>
</table>

**entity:AudioAreaRandom node**

Used to enable and disable the associated entity.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Disables the entity</td>
</tr>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Enables the entity</td>
</tr>
</tbody>
</table>

**PreloadData node**

Used to load and unload preload requests to optimize memory consumption. This node lists only preloads that are not set to **Autoload** in the Audio Controls Editor.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preload Request</td>
<td>String</td>
<td>Defines the preload requests that should be loaded or unloaded</td>
</tr>
<tr>
<td>Load</td>
<td>Any</td>
<td>Loads the preload requests</td>
</tr>
<tr>
<td>Unload</td>
<td>Any</td>
<td>Unloads the preload requests</td>
</tr>
</tbody>
</table>

Rtpc node

Use to change RTPC values. If you have an entity assigned to this node, the RTPC assigned to the Name input controls parameters only on the assigned entity. If no entity is assigned, the parameter change is applied to all entities.

For Wwise, any RTPC that is not assigned to an entity sets connected game parameters on all game objects. An RTPC that is assigned to an entity sets the connected game parameters only on the game object corresponding to the assigned entity in Wwise. You can monitor the RTPC changes for an entity in the game object profiler layout.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td>Name of the RTPC</td>
</tr>
<tr>
<td>Value</td>
<td>Float</td>
<td>Sets the RTPC value</td>
</tr>
</tbody>
</table>

Switch node

Used to set the state of a switch. Multiple states can be selected in the node to reduce the complexity of flow graph logic when more than one state change should happen.

For Wwise, a connected switch state sets the Wwise switch only on a game object corresponding to the assigned entity. A switch state connected to a Wwise switch without an assigned entity is set on the Dummy Game object in Wwise. A switch state connected to a Wwise state always sets the state globally, regardless of the assigned entity.
Audio Nodes

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch</td>
<td>String</td>
<td>Switch name</td>
</tr>
<tr>
<td>State1 - State4</td>
<td>String</td>
<td>Name of the state</td>
</tr>
<tr>
<td>SetState1 - SetState4</td>
<td>Any</td>
<td>Sets the state</td>
</tr>
</tbody>
</table>

Trigger node

Used to trigger events.

For Wwise, a trigger without an entity assigned is executed on the dummy game object in Wwise. A trigger with an entity assigned is executed on the game object corresponding to the assigned entity.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PlayTrigger</td>
<td>String</td>
<td>The name of the event. Any event can be triggered with this node.</td>
</tr>
<tr>
<td>StopTrigger</td>
<td>String</td>
<td>The name of the event. Any event can be triggered with this node. If no event is defined and a sound is started on the corresponding PlayTrigger, it stops at once when the stop input is triggered.</td>
</tr>
<tr>
<td>Play</td>
<td>Any</td>
<td>Triggers the event defined in the PlayTrigger input.</td>
</tr>
<tr>
<td>Stop</td>
<td>Any</td>
<td>Triggers the event defined in the StopTrigger input.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when the event is complete</td>
</tr>
</tbody>
</table>
Camera Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to configure player camera settings.

**Topics**
- GetTransform node (p. 281)
- View node (p. 281)
- ViewShakeEx node (p. 282)

**GetTransform node**

Used to get and output the position and direction of the player camera.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get</td>
<td>Any</td>
<td>Triggers the retrieval of the currently active camera position and direction</td>
</tr>
</tbody>
</table>

**View node**

Used to create a custom view linked to the chosen entity.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>Vec3</td>
<td>Outputs camera position</td>
</tr>
<tr>
<td>Dir</td>
<td>Vec3</td>
<td>Outputs camera direction</td>
</tr>
</tbody>
</table>
### Camera Nodes

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Deactivates the node</td>
</tr>
<tr>
<td>FOV</td>
<td>Float</td>
<td>Camera field of view</td>
</tr>
<tr>
<td>Blend</td>
<td>Boolean</td>
<td>Whether to blend the camera or not</td>
</tr>
<tr>
<td>BlendFOVSpeed</td>
<td>Float</td>
<td>Blended field of view speed</td>
</tr>
<tr>
<td>BlendFOVOffset</td>
<td>Float</td>
<td>Blended field of view offset</td>
</tr>
<tr>
<td>BlendPosSpeed</td>
<td>Float</td>
<td>Blended position speed</td>
</tr>
<tr>
<td>BlendPosOffset</td>
<td>Vec3</td>
<td>Blended position offset</td>
</tr>
<tr>
<td>BlendRotSpeed</td>
<td>Float</td>
<td>Blended rotation speed</td>
</tr>
<tr>
<td>BlendRotOffset</td>
<td>Vec3</td>
<td>Blended rotation offset</td>
</tr>
</tbody>
</table>

#### ViewShakeEx node

Used to enable camera shake on the player's view. You can specify the fade in and out durations and stop the effect.

```
CameraViewShakeEx
  | trigger
  | restrict=None
  | view=firstperson
  | groundonly=false
  | smoothfalse
  | angle=[0,0,0,0,0,0]
  | shift=[0,0,0,0,0,0]
  | frequency=10
  | randomness=1
  | distance=0
  | rangeMin=10
  | rangeMax=30
  | sustainDuration=0
  | fadeInDuration=9
  | fadeOutDuration=3
  | stop
  | preset=nbPreset
```

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>Restrict</td>
<td>Integer</td>
<td>Restricts the view</td>
</tr>
<tr>
<td>View</td>
<td>Integer</td>
<td>Which camera view to use</td>
</tr>
<tr>
<td>GroundOnly</td>
<td>Boolean</td>
<td>Applies shake only when the player is standing on the ground</td>
</tr>
<tr>
<td>Smooth</td>
<td>Boolean</td>
<td>Any sudden direction changes</td>
</tr>
<tr>
<td>Angle</td>
<td>Vec3</td>
<td>Shake angle</td>
</tr>
</tbody>
</table>
### ComponentEntity Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to get and set various component entity system settings. These nodes only work with the component entity system.

In addition, with the following exceptions, flow graph nodes that have an Entity ID input port do not work with the component entity system nodes:

- **Physics:ActionImpulse** node
- **Physics:Dynamics** node
- **Movement:RotateEntity** node
- **Entity:EntityID** node

#### Topics

- Audio:ExecuteOneShot node (p. 284)
- Audio:StopOneShot node (p. 284)
- EventActionHandler:AZVector3 node (p. 284)
- EventActionHandler:EntityID node (p. 285)
- EventActionHandler:Float node (p. 285)
- EventActionSender:AZVector3 node (p. 286)

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift</td>
<td>Vec3</td>
<td>Shake shift</td>
</tr>
<tr>
<td>Frequency</td>
<td>Float</td>
<td>Shake frequency</td>
</tr>
<tr>
<td>Randomness</td>
<td>Float</td>
<td>Randomness of shake</td>
</tr>
<tr>
<td>Distance</td>
<td>Float</td>
<td>Distance to effect source</td>
</tr>
<tr>
<td>RangeMin</td>
<td>Float</td>
<td>Minimum strength effect range</td>
</tr>
<tr>
<td>RangeMax</td>
<td>Float</td>
<td>Maximum strength effect range</td>
</tr>
<tr>
<td>SustainDuration</td>
<td>Float</td>
<td>Duration of the non-fading part of the shake</td>
</tr>
<tr>
<td>FadeInDuration</td>
<td>Float</td>
<td>Fade in time</td>
</tr>
<tr>
<td>FadeOutDuration</td>
<td>Float</td>
<td>Fade out time</td>
</tr>
<tr>
<td>Stop</td>
<td>Any</td>
<td>Stops the shake</td>
</tr>
<tr>
<td>Preset</td>
<td>Integer</td>
<td>Preset input values</td>
</tr>
</tbody>
</table>
• EventActionSender:EntityID node (p. 286)
• EventActionSender:Float node (p. 287)
• GameplayEventHandler:AZVector3 node (p. 287)
• GameplayEventHandler:EntityID node (p. 288)
• GameplayEventHandler:Float node (p. 288)
• GameplayEventSender:AZVector3 node (p. 289)
• GameplayEventSender:EntityID node (p. 289)
• GameplayEventSender:Float node (p. 289)
• Light:Switch node (p. 290)
• Particles:Switch node (p. 290)
• TransformComponent:GetEntityPosition node (p. 290)
• TransformComponent:GetEntityRotation node (p. 291)
• TransformComponent:SetEntityPosition node (p. 291)
• TransformComponent:SetEntityRotation node (p. 292)
• TriggerComponent:EnterTrigger node (p. 292)

Audio:ExecuteOneShot node

Used to execute the audio trigger as a one-shot on the entity.

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
</tr>
<tr>
<td>Trigger</td>
<td>String</td>
</tr>
</tbody>
</table>

Audio:StopOneShot node

Used to stop the specified audio one shot trigger.

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
</tr>
<tr>
<td>Trigger</td>
<td>String</td>
</tr>
</tbody>
</table>

EventActionHandler:AZVector3 node

Used for the entity event action handler.
ComponentEntity Nodes

**Input**

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChannelId</td>
<td>Any</td>
</tr>
<tr>
<td>EventName</td>
<td>String</td>
</tr>
<tr>
<td>Enable</td>
<td>Any</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
</tr>
</tbody>
</table>

**Output**

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Vec3 Vector value on event action handler success</td>
</tr>
<tr>
<td>Failure</td>
<td>Vec3 Vector value on event action handler failure</td>
</tr>
</tbody>
</table>

**EventActionHandler:EntityID node**

Used for the entity event action handler.

**Input**

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChannelId</td>
<td>Any</td>
</tr>
<tr>
<td>EventName</td>
<td>String</td>
</tr>
<tr>
<td>Enable</td>
<td>Any</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
</tr>
</tbody>
</table>

**Output**

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any Value on event action handler success</td>
</tr>
<tr>
<td>Failure</td>
<td>Any Value on event action handler failure</td>
</tr>
</tbody>
</table>

**EventActionHandler:Float node**

Used for the entity event action handler.
**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChannelId</td>
<td>Any</td>
</tr>
<tr>
<td>EventName</td>
<td>String</td>
</tr>
<tr>
<td>Enable</td>
<td>Any</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Float value on event action handler success</td>
</tr>
<tr>
<td>Failure</td>
<td>Float value on event action handler failure</td>
</tr>
</tbody>
</table>

**EventActionSender:AZVector3 node**

Used for the entity event action sender.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
</tr>
<tr>
<td>ChannelID</td>
<td>Any</td>
</tr>
<tr>
<td>SendEventValue</td>
<td>Vec3</td>
</tr>
<tr>
<td>Eventname</td>
<td>String</td>
</tr>
</tbody>
</table>

**EventActionSender:EntityID node**

Used for the entity event action sender.
**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
</tr>
<tr>
<td>ChannelID</td>
<td>Any</td>
</tr>
<tr>
<td>SendEventValue</td>
<td>Any</td>
</tr>
<tr>
<td>Eventname</td>
<td>String</td>
</tr>
</tbody>
</table>

**EventActionSender:Float node**

Used for the entity event action sender.

**GameplayEventHandler:AZVector3 node**

Used for the gameplay event handler.
Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Vec3 Vector value on event gameplay event handler success</td>
</tr>
<tr>
<td>Failure</td>
<td>Vec3 Vector value on gameplay event handler failure</td>
</tr>
</tbody>
</table>

GameplayEventHandler:EntityID node

Used for the gameplay event handler.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChannelId</td>
<td>Any Entity channel ID</td>
</tr>
<tr>
<td>EventName</td>
<td>String Name of the gameplay event handler</td>
</tr>
<tr>
<td>Enable</td>
<td>Any Enables the gameplay event handler</td>
</tr>
<tr>
<td>Disable</td>
<td>Any Disables the gameplay event handler</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any Value on gameplay event handler success</td>
</tr>
<tr>
<td>Failure</td>
<td>Any Value on event gameplay event handler failure</td>
</tr>
</tbody>
</table>

GameplayEventHandler:Float node

Used for the gameplay event handler.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChannelId</td>
<td>Any Entity channel ID</td>
</tr>
<tr>
<td>EventName</td>
<td>String Name of the gameplay event handler</td>
</tr>
<tr>
<td>Enable</td>
<td>Any Enables the gameplay event handler</td>
</tr>
<tr>
<td>Disable</td>
<td>Any Disables the gameplay event handler</td>
</tr>
</tbody>
</table>
Lumberyard Legacy Reference
ComponentEntity Nodes

## Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Float value on event action handler success</td>
</tr>
<tr>
<td>Failure</td>
<td>Float value on event action handler failure</td>
</tr>
</tbody>
</table>

### GameplayEventSender:AZVector3 node

Used for the gameplay event sender.

```plaintext
ComponentEntity.GameplayEventSender AZVector3
  # Activate
  # ChannelID
  # SendEventValue=0,0,0
  # EventName=
```

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
</tr>
<tr>
<td>ChannelID</td>
<td>Any</td>
</tr>
<tr>
<td>SendEventValue</td>
<td>Vec3</td>
</tr>
<tr>
<td>Eventname</td>
<td>String</td>
</tr>
</tbody>
</table>

### GameplayEventSender:EntityID node

Used for the gameplay event sender.

```plaintext
ComponentEntity.GameplayEventSender EntityID
  # Activate
  # ChannelID
  # SendEventValue
  # EventName=
```

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
</tr>
<tr>
<td>ChannelID</td>
<td>Any</td>
</tr>
<tr>
<td>SendEventValue</td>
<td>Any</td>
</tr>
<tr>
<td>Eventname</td>
<td>String</td>
</tr>
</tbody>
</table>

### GameplayEventSender:Float node

Used for the gameplay event sender.

```plaintext
ComponentEntity.GameplayEventSender Float
  # Activate
  # ChannelID
  # SendEventValue=0
  # EventName=
```

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
</tr>
<tr>
<td>ChannelID</td>
<td>Any</td>
</tr>
<tr>
<td>SendEventValue</td>
<td>Float</td>
</tr>
<tr>
<td>Eventname</td>
<td>String</td>
</tr>
</tbody>
</table>
## ComponentEntity Nodes

### Activate

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
</tr>
</tbody>
</table>

Activates the node.

### ChannelID

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChannelID</td>
<td>Any</td>
</tr>
</tbody>
</table>

ID of the channel for the gameplay event sender.

### SendEventValue

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SendEventValue</td>
<td>Float</td>
</tr>
</tbody>
</table>

Float value for the gameplay event sender.

### Eventname

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eventname</td>
<td>String</td>
</tr>
</tbody>
</table>

Name of the gameplay event sender.

---

### Light:Switch node

Used to turn the light entity on or off.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>On</td>
<td>Any</td>
</tr>
<tr>
<td>Off</td>
<td>Any</td>
</tr>
</tbody>
</table>

Turns the light on/off.

---

### Particles:Switch node

Used to show or hide the particle entity.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show</td>
<td>Any</td>
</tr>
<tr>
<td>Hide</td>
<td>Any</td>
</tr>
</tbody>
</table>

Displays/Hides the particle.

---

### TransformComponent:GetEntityPosition node

Used to get the entity position.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
</tr>
</tbody>
</table>

Activates the node.
### ComponentEntity Nodes

#### Port Description

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coordinate System</td>
<td>Integer</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CurrentPosition</td>
<td>Vec3</td>
</tr>
</tbody>
</table>

### TransformComponent:GetEntityRotation node

**Used to get the entity rotation.**

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
</tr>
<tr>
<td>Coordinate System</td>
<td>Integer</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CurrentRotation</td>
<td>Vec3</td>
</tr>
<tr>
<td>Forward</td>
<td>Vec3</td>
</tr>
<tr>
<td>Up</td>
<td>Vec3</td>
</tr>
<tr>
<td>Right</td>
<td>Vec3</td>
</tr>
</tbody>
</table>

### TransformComponent:SetEntityPosition node

**Used to set entity position.**

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
</tr>
<tr>
<td>NewPosition</td>
<td>Any</td>
</tr>
<tr>
<td>Coordinate System</td>
<td>Integer</td>
</tr>
</tbody>
</table>

#### Outputs
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
</tr>
<tr>
<td>NewPosition</td>
<td>Vec3</td>
</tr>
<tr>
<td>Coordinate System</td>
<td>Integer</td>
</tr>
</tbody>
</table>

**TransformComponent:SetEntityRotation node**

Used to set entity rotation.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
</tr>
<tr>
<td>Rotation</td>
<td>Vec3</td>
</tr>
<tr>
<td>Coordinate System</td>
<td>Integer</td>
</tr>
</tbody>
</table>

**TriggerComponent:EnterTrigger node**

Used to trigger when the entity enters or leaves the trigger area.

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entered</td>
<td>Triggers when entity enters the trigger area</td>
</tr>
<tr>
<td>Exited</td>
<td>Triggers when entity leaves the trigger area</td>
</tr>
</tbody>
</table>

**CustomAction Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.
You can use the following flow graph nodes to control custom actions that entities take.

**Topics**
- Abort node (p. 293)
- Control node (p. 293)
- End node (p. 294)
- Start node (p. 295)
- Succeed node (p. 295)
- SucceedWait node (p. 295)
- SucceedWaitComplete node (p. 295)

**Abort node**
Used to start the abort path of a custom action.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectId</td>
<td>Any</td>
<td>Entity ID of the object on which the custom action is executing on</td>
</tr>
</tbody>
</table>

**Control node**
Used to control a custom action instance.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>String</td>
<td>Entity is entering the start path</td>
</tr>
<tr>
<td>Succeed</td>
<td>String</td>
<td>Entity is entering the succeed path</td>
</tr>
<tr>
<td>SucceedWait</td>
<td>String</td>
<td>Entity is entering the succeed wait path</td>
</tr>
<tr>
<td>SucceedWaitComplete</td>
<td>String</td>
<td>Entity is entering the succeed wait complete path</td>
</tr>
<tr>
<td>Abort</td>
<td>Any</td>
<td>Entity is entering the abort path</td>
</tr>
<tr>
<td>EndSuccess</td>
<td>Any</td>
<td>Entity is entering the end succeed path</td>
</tr>
</tbody>
</table>
### CustomAction Nodes

#### End Failure

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EndFailure</td>
<td>Any</td>
<td>Entity is entering the end failure path</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Started</td>
<td>Any</td>
<td>Entity has entered the start path</td>
</tr>
<tr>
<td>Succeeded</td>
<td>Any</td>
<td>Entity has entered the succeed path</td>
</tr>
<tr>
<td>SucceedWait</td>
<td>Any</td>
<td>Entity has entered the succeed wait path</td>
</tr>
<tr>
<td>SucceedWaitComplete</td>
<td>Any</td>
<td>Entity has entered the succeed wait completed</td>
</tr>
<tr>
<td>Aborted</td>
<td>Any</td>
<td>Entity has entered the abort path</td>
</tr>
<tr>
<td>EndedSuccess</td>
<td>Any</td>
<td>Entity has entered the end success path</td>
</tr>
<tr>
<td>EndedFailure</td>
<td>Any</td>
<td>Entity has entered the end failure path</td>
</tr>
</tbody>
</table>

#### End node

Used to end a custom action.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Succeed</td>
<td>Any</td>
<td>Entity has entered the succeed path</td>
</tr>
<tr>
<td>SucceedWait</td>
<td>Any</td>
<td>Entity has entered the succeed wait path</td>
</tr>
<tr>
<td>SucceedWaitComplete</td>
<td>Any</td>
<td>Entity has entered the succeed wait complete path</td>
</tr>
<tr>
<td>Abort</td>
<td>Any</td>
<td>Entity has entered the abort path</td>
</tr>
<tr>
<td>EndSuccess</td>
<td>Any</td>
<td>Entity has entered the end succeed path</td>
</tr>
</tbody>
</table>
### CustomAction Nodes

#### Port | Type | Description
---|---|---
EndFailure | Any | Entity has entered the end failure path

### Start node

Used to start a custom action.

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectId</td>
<td>Any</td>
<td>Entity ID of the object on which the custom action is executing on</td>
</tr>
</tbody>
</table>

### Succeed node

Used to indicate a custom action succeeded.

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectId</td>
<td>Any</td>
<td>Entity ID of the object on which the custom action is executing on</td>
</tr>
</tbody>
</table>

### SucceedWait node

Used to indicate that a custom action wait succeeded.

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectId</td>
<td>Any</td>
<td>Entity ID of the object on which the custom action is executing on</td>
</tr>
</tbody>
</table>

### SucceedWaitComplete node

Used to indicate that a custom action wait succeeded and completed.

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectId</td>
<td>Any</td>
<td>Entity ID of the object on which the custom action is executing on</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectId</td>
<td>Any</td>
<td>Entity ID of the object on which the custom action is executing on</td>
</tr>
</tbody>
</table>

Debug Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to configure various settings used for debugging purposes.

Topics

- CSVDumper node (p. 296)
- ConsoleVariable node (p. 297)
- DisplayMessage node (p. 297)
- Draw2d nodes (p. 298)
- Draw nodes (p. 300)
- ExecuteString node (p. 305)
- FloatToString node (p. 305)
- Frame node (p. 306)
- FrameExtended node (p. 306)
- InputKey node (p. 307)
- Log node (p. 307)
- Memory node (p. 307)

CSVDumper node

Used to store the cell values of the specified .csv file.
Debug Nodes

### ConsoleVariable node

Used to set and get the value of a console variable.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>Any</td>
<td>Set console variable value</td>
</tr>
<tr>
<td>Get</td>
<td>Any</td>
<td>Get console variable value</td>
</tr>
<tr>
<td>CVar</td>
<td>String</td>
<td>Name of console variable</td>
</tr>
<tr>
<td>Value</td>
<td>String</td>
<td>Value of console variable to set</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CurValue</td>
<td></td>
<td>Current value of the console variable</td>
</tr>
</tbody>
</table>

DisplayMessage node

If an entity is not provided, the local player will be used instead.
Lumberyard Legacy Reference
Debug Nodes

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show</td>
<td>Any</td>
<td>Show message</td>
</tr>
<tr>
<td>Hide</td>
<td>Any</td>
<td>Hide message</td>
</tr>
<tr>
<td>message</td>
<td>String</td>
<td>Message to display on the HUD</td>
</tr>
<tr>
<td>DisplayTime</td>
<td>FloatFloat</td>
<td>Duration that the message will be visible for</td>
</tr>
<tr>
<td>posx</td>
<td>Float</td>
<td>Input x text position</td>
</tr>
<tr>
<td>posy</td>
<td>Float</td>
<td>Input y text position</td>
</tr>
<tr>
<td>fontSize</td>
<td>Float</td>
<td>Input font size</td>
</tr>
<tr>
<td>color</td>
<td>Vec3</td>
<td>Color of the message text</td>
</tr>
<tr>
<td>centered</td>
<td>Boolean</td>
<td>Centers the text around the coordinates</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show</td>
<td>Any</td>
<td>Displays the message</td>
</tr>
<tr>
<td>Hide</td>
<td>Any</td>
<td>Hides the message</td>
</tr>
</tbody>
</table>

Draw2d nodes

Draw2d:Circle node

Used to draw a circle.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw</td>
<td>Any</td>
<td>Draws a 2D circle</td>
</tr>
</tbody>
</table>
### Port | Type | Description
-------|------|----------------------------------
ScreenX | Float | X-axis position of the center of the circle
ScreenY | Float | Y-axis position of the center of the circle
Radius  | Float | Radius of the circle
Color   | Vec3  | Color of the circle
Opacity | Float | Transparency of the circle
Time    | Float | Number of seconds the circle will be visible for

#### Draw2d:Line node
Used to draw a line.

![Debug:Draw2d:Line](image)

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw</td>
<td>Any</td>
<td>Draws a line</td>
</tr>
<tr>
<td>StartX</td>
<td>Float</td>
<td>X-axis starting point of the line</td>
</tr>
<tr>
<td>StartY</td>
<td>Float</td>
<td>Y-axis starting point of the line</td>
</tr>
<tr>
<td>EndX</td>
<td>Float</td>
<td>X-axis ending point of the line</td>
</tr>
<tr>
<td>EndY</td>
<td>Float</td>
<td>Y-axis ending point of the line</td>
</tr>
<tr>
<td>Color</td>
<td>Vec3</td>
<td>Color of the line</td>
</tr>
<tr>
<td>Opacity</td>
<td>Float</td>
<td>Transparency of the line</td>
</tr>
<tr>
<td>Time</td>
<td>Float</td>
<td>Number of seconds the line will be visible for</td>
</tr>
</tbody>
</table>

#### Draw2d:Rectangle node
Used to draw a rectangle.

![Debug:Draw2d:Rectangle](image)
### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw</td>
<td>Any</td>
<td>Draws a rectangle</td>
</tr>
<tr>
<td>ScreenX</td>
<td>Float</td>
<td>X-axis position of the center of the rectangle</td>
</tr>
<tr>
<td>ScreenY</td>
<td>Float</td>
<td>X-axis position of the center of the rectangle</td>
</tr>
<tr>
<td>Width</td>
<td>Float</td>
<td>Width of the rectangle</td>
</tr>
<tr>
<td>Height</td>
<td>Float</td>
<td>Height of the rectangle</td>
</tr>
<tr>
<td>Centered</td>
<td>Boolean</td>
<td>Rectangle centered at ScreenX and ScreenY</td>
</tr>
<tr>
<td>Color</td>
<td>Vec3</td>
<td>Color of the rectangle</td>
</tr>
<tr>
<td>Opacity</td>
<td>Float</td>
<td>Transparency of the rectangle</td>
</tr>
<tr>
<td>Time</td>
<td>Float</td>
<td>Number of seconds the rectangle will be visible for</td>
</tr>
</tbody>
</table>

### Draw2d:Text node

Used to output a text message.

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw</td>
<td>Any</td>
<td>Displays text</td>
</tr>
<tr>
<td>Text</td>
<td>String</td>
<td>Text to display</td>
</tr>
<tr>
<td>ScreenX</td>
<td>Float</td>
<td>X-axis position of the text</td>
</tr>
<tr>
<td>ScreenY</td>
<td>Float</td>
<td>Y-axis position of the text</td>
</tr>
<tr>
<td>FontSize</td>
<td>Float</td>
<td>Text message font size</td>
</tr>
<tr>
<td>Color</td>
<td>Vec3</td>
<td>Color of the text</td>
</tr>
<tr>
<td>Opacity</td>
<td>Float</td>
<td>Transparency of the text</td>
</tr>
<tr>
<td>Time</td>
<td>Float</td>
<td>Number of seconds the text will be visible for</td>
</tr>
</tbody>
</table>

### Draw nodes

**Draw:AABB node**

Used to draw an AABB bounding box.
Debug Nodes

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw</td>
<td>Any</td>
<td>Draws an AABB bounding box</td>
</tr>
<tr>
<td>MinPos</td>
<td>Vec3</td>
<td>Minimum position of the bounding box</td>
</tr>
<tr>
<td>MaxPos</td>
<td>Vec3</td>
<td>Maximum position of the bounding box</td>
</tr>
<tr>
<td>Color</td>
<td>Vec3</td>
<td>Color of the bounding box</td>
</tr>
<tr>
<td>Time</td>
<td>Float</td>
<td>Number of seconds the bounding box will be visible for</td>
</tr>
</tbody>
</table>

Draw:Cone node

Used a draw a cone.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw</td>
<td>Any</td>
<td>Draws a cone</td>
</tr>
<tr>
<td>Pos</td>
<td>Vec3</td>
<td>Position of the cone</td>
</tr>
<tr>
<td>Dir</td>
<td>Vec3</td>
<td>Direction of the cone axis</td>
</tr>
<tr>
<td>Radius</td>
<td>Float</td>
<td>Radius of the cone base</td>
</tr>
<tr>
<td>Height</td>
<td>Float</td>
<td>Height of the cone</td>
</tr>
<tr>
<td>Color</td>
<td>Vec3</td>
<td>Color of the cone</td>
</tr>
<tr>
<td>Time</td>
<td>Float</td>
<td>Number of seconds the cone will be visible for</td>
</tr>
</tbody>
</table>

Draw:Cylinder node

Used to draw a cylinder.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw</td>
<td>Any</td>
<td>Draws a cylinder</td>
</tr>
<tr>
<td>Pos</td>
<td>Vec3</td>
<td>Position of the cylinder</td>
</tr>
<tr>
<td>Dir</td>
<td>Vec3</td>
<td>Direction of the cylinder axis</td>
</tr>
<tr>
<td>Radius</td>
<td>Float</td>
<td>Radius of the cylinder</td>
</tr>
<tr>
<td>Height</td>
<td>Float</td>
<td>Height of the cylinder</td>
</tr>
<tr>
<td>Color</td>
<td>Vec3</td>
<td>Color of the cylinder</td>
</tr>
<tr>
<td>Time</td>
<td>Float</td>
<td>Number of seconds the cylinder will be visible for</td>
</tr>
</tbody>
</table>
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw</td>
<td>Any</td>
<td>Draws a cylinder</td>
</tr>
<tr>
<td>Pos</td>
<td>Vec3</td>
<td>Position of the cylinder</td>
</tr>
<tr>
<td>Dir</td>
<td>Vec3</td>
<td>Direction of the cylinder axis</td>
</tr>
<tr>
<td>Radius</td>
<td>Float</td>
<td>Radius of the cylinder</td>
</tr>
<tr>
<td>Height</td>
<td>Float</td>
<td>Height of the cylinder</td>
</tr>
<tr>
<td>Color</td>
<td>Vec3</td>
<td>Color of the cylinder</td>
</tr>
<tr>
<td>Time</td>
<td>Float</td>
<td>Number of seconds the cylinder will be visible for</td>
</tr>
</tbody>
</table>

**Draw:Direction node**

Used to draw an arrow.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw</td>
<td>Any</td>
<td>Draws an arrow</td>
</tr>
<tr>
<td>Pos</td>
<td>Vec3</td>
<td>Position of the arrow</td>
</tr>
<tr>
<td>Dir</td>
<td>Vec3</td>
<td>Direction the arrow is pointing</td>
</tr>
<tr>
<td>Radius</td>
<td>Float</td>
<td>Radius of the arrow head</td>
</tr>
<tr>
<td>Color</td>
<td>Vec3</td>
<td>Color of the arrow</td>
</tr>
<tr>
<td>Time</td>
<td>Float</td>
<td>Number of seconds the arrow will be visible for</td>
</tr>
</tbody>
</table>

**Draw:EntityTag node**

Used to draw a text message above an entity.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw</td>
<td>Any</td>
<td>Displays a text message above an entity</td>
</tr>
</tbody>
</table>
### Debug Nodes

#### Port | Type | Description
--- | --- | ---
Message | String | Text message
FontSize | Float | Text message font size
Color | Vec3 | Text message color
Time | Float | Number of seconds the message will be visible for

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Done |  | Triggers when the text message is no longer visible

#### Draw:EntityTagAdvanced node

Used to draw a text message above an entity.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Draw | Any | Displays a text message above an entity
| Message | String | Message to be displayed
| FadeTime | Float | Number of seconds for text message to fade out
| FontSize | Float | Font size of the text message
| ViewDistance | Float | Distance from camera the entity must be within for message to be displayed
| StaticID | String | Static tag ID
| ColumnNum | Integer | Which column above an entity the message will be displayed in
| Color | Vec3 | Color of the text message
| Time | Float | Number of seconds the text message will be visible for

---

303
Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when the text message is no longer visible</td>
</tr>
</tbody>
</table>

**Draw:Line node**

Used to draw a line.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw</td>
<td>Any</td>
<td>Draws a line in 3D space</td>
</tr>
<tr>
<td>Pos1</td>
<td>Vec3</td>
<td>Starting point of the line</td>
</tr>
<tr>
<td>Pos2</td>
<td>Vec3</td>
<td>Ending point of the line</td>
</tr>
<tr>
<td>Dir</td>
<td>Vec3</td>
<td>Direction of the line</td>
</tr>
<tr>
<td>Length</td>
<td>Float</td>
<td>Length of the line</td>
</tr>
<tr>
<td>Color</td>
<td>Vec3</td>
<td>Color of the line</td>
</tr>
<tr>
<td>Time</td>
<td>Float</td>
<td>Number of seconds the circle will be visible for</td>
</tr>
</tbody>
</table>

**Draw:PlanarDisc node**

Used to draw a disc.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw</td>
<td>Any</td>
<td>Draws a disc</td>
</tr>
<tr>
<td>Pos</td>
<td>Vec3</td>
<td>Position of the disc center</td>
</tr>
<tr>
<td>InnerRadius</td>
<td>Float</td>
<td>Inner radius of the disc</td>
</tr>
<tr>
<td>OuterRadius</td>
<td>Float</td>
<td>Outer radius of the disc</td>
</tr>
<tr>
<td>Color</td>
<td>Vec3</td>
<td>Color of the disc</td>
</tr>
<tr>
<td>Time</td>
<td>Float</td>
<td>Number of seconds the circle will be visible for</td>
</tr>
</tbody>
</table>
Draw: Sphere node

Used to draw a sphere.

```
Draw
Pos=0,0,0
Radius=0
Color=0,0,0
Time=0
```

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw</td>
<td>Any</td>
<td>Draws a sphere</td>
</tr>
<tr>
<td>Pos</td>
<td>Vec3</td>
<td>Position of the sphere center</td>
</tr>
<tr>
<td>Radius</td>
<td>Float</td>
<td>Radius of the sphere</td>
</tr>
<tr>
<td>Color</td>
<td>Vec3</td>
<td>Color of the sphere</td>
</tr>
<tr>
<td>Time</td>
<td>Float</td>
<td>Number of seconds the circle will be visible for</td>
</tr>
</tbody>
</table>

ExecuteString node

Used to execute a string when using the console.

```
Set
String=
NextFrame=0
```

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>Any</td>
<td>Executes the string</td>
</tr>
<tr>
<td>String</td>
<td>String</td>
<td>String to be executed</td>
</tr>
<tr>
<td>NextFrame</td>
<td>Boolean</td>
<td>String will be executed next frame</td>
</tr>
</tbody>
</table>

FloatToString node

Used to output a float value in string format with a limited number of decimals.

```
Activate
Number=0
AmountOfDecimals=0
```

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Number</td>
<td>Float</td>
<td>Floating point number to convert</td>
</tr>
<tr>
<td>AmountOfDecimals</td>
<td>Integer</td>
<td>Number of decimal places for the floating point</td>
</tr>
</tbody>
</table>
### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>String</td>
<td>Outputs a string representation of the floating point input</td>
</tr>
</tbody>
</table>

### Frame node

Used to output the current frame rate data.

- **frametime**: Current frame time
- **framerate**: Current frame rate
- **frameid**: Frame ID

### FrameExtended node

Used to output extended current frame rate data.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Start collecting frame rate data</td>
</tr>
<tr>
<td>Stop</td>
<td>Any</td>
<td>Stop collecting frame rate data</td>
</tr>
<tr>
<td>Reset</td>
<td>Any</td>
<td>Resets the data</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FrameTime</td>
<td>Float</td>
<td>Current frame time</td>
</tr>
<tr>
<td>FrameRate</td>
<td>Float</td>
<td>Current frame rate</td>
</tr>
<tr>
<td>FrameId</td>
<td>Integer</td>
<td>Frame ID</td>
</tr>
<tr>
<td>MinFrameRate</td>
<td>Float</td>
<td>Minimum frame rate</td>
</tr>
</tbody>
</table>
## InputKey node

Used to catch key inputs. The Entity input is required for multiplayer games.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Deactivates the node</td>
</tr>
<tr>
<td>Key</td>
<td>String</td>
<td>Key name</td>
</tr>
<tr>
<td>NonDevMode</td>
<td>String</td>
<td>Can be used in non-dev mode if set to true</td>
</tr>
<tr>
<td>Keyboard only</td>
<td>Boolean</td>
<td>Ignores non-keyword data if set to true</td>
</tr>
</tbody>
</table>

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pressed</td>
<td>String</td>
<td>Triggers when a key is pressed</td>
</tr>
<tr>
<td>Released</td>
<td>String</td>
<td>Triggers when a key is released</td>
</tr>
</tbody>
</table>

## Log node

Used to log string input messages to the console.

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>message</td>
<td>String</td>
<td>Message to be logged</td>
</tr>
</tbody>
</table>

## Memory node

Used to display video memory data.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysmem</td>
<td>Integer</td>
<td>Outputs system video memory data</td>
</tr>
<tr>
<td>videomem_thisframe</td>
<td>Integer</td>
<td>Outputs video memory used for current frame</td>
</tr>
<tr>
<td>videomem_recently</td>
<td>Integer</td>
<td>Outputs video memory recently used</td>
</tr>
<tr>
<td>meshmem</td>
<td>Integer</td>
<td>Outputs memory used for the mesh object</td>
</tr>
</tbody>
</table>

**Dialog Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard’s new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to configure and control actor dialogs.

**Note**

These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at Lumberyard Downloads.

**Topics**

- PlayDialog node (p. 308)

**PlayDialog node**

Used to play a dialog.
## Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Play</td>
<td>Any</td>
<td>Plays the dialog</td>
</tr>
<tr>
<td>Stop</td>
<td>Any</td>
<td>Stops the dialog</td>
</tr>
<tr>
<td>Dialog</td>
<td>String</td>
<td>Name of the dialog to play</td>
</tr>
<tr>
<td>StartLine</td>
<td>Integer</td>
<td>Line to start the dialog from</td>
</tr>
<tr>
<td>AIInterrupt</td>
<td>Integer</td>
<td>AI interrupt behavior; values are Never, Alert, and Combat</td>
</tr>
<tr>
<td>AwareDistance</td>
<td>Float</td>
<td>Distance that player is considered as listening at</td>
</tr>
<tr>
<td>AwareAngle</td>
<td>Float</td>
<td>View angle that player is considered as listening at</td>
</tr>
<tr>
<td>AwareTimeout</td>
<td>Float</td>
<td>Time out until non-aware player aborts dialog</td>
</tr>
<tr>
<td>Flags</td>
<td>Integer</td>
<td>Dialog playback flags</td>
</tr>
<tr>
<td>Buffer</td>
<td>String</td>
<td>Stores the dialog. Only one dialog can be played at any time in each buffer</td>
</tr>
<tr>
<td>BufferDisplay</td>
<td>Float</td>
<td>How many more seconds the dialog will wait until the previous dialog in its dialog has finished</td>
</tr>
<tr>
<td>Actor 1-8</td>
<td>Any</td>
<td>Actor entity IDs</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Started</td>
<td>Any</td>
<td>Triggered when the dialog has started</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when the dialog has finished or aborted</td>
</tr>
<tr>
<td>Finished</td>
<td>Any</td>
<td>Triggered when the dialog has finished</td>
</tr>
<tr>
<td>Aborted</td>
<td>Any</td>
<td>Triggered when the dialog has aborted</td>
</tr>
<tr>
<td>PlayerAbort</td>
<td>Integer</td>
<td>Triggered when the dialog has aborted because the player is out of range or out of view</td>
</tr>
<tr>
<td>AIAbort</td>
<td>Any</td>
<td>Triggered when the dialog has aborted because the AI got alerted</td>
</tr>
</tbody>
</table>
Dynamic Response Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard’s new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to configure settings for the Dynamic Response system.

**Note**
These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at Lumberyard Downloads.

**Topics**
- SendSignal node (p. 310)
- SetFloatVariable node (p. 311)
- SetIntegerVariable node (p. 312)
- SetStringVariable node (p. 312)

### SendSignal node

Used to send a signal to the Dynamic Response system.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send</td>
<td>Any</td>
<td>Sends the dynamic response signal</td>
</tr>
<tr>
<td>Cancel</td>
<td>Any</td>
<td>Cancels the dynamic response signal</td>
</tr>
</tbody>
</table>

---

### Port Type Description

<p>| ActorDied | Any | Triggered when the dialog has aborted because the Actor died               |
| LastLine  | Integer | Last line played when the dialog was aborted                               |
| CurLine   | Integer | Current line; triggered whenever a line starts                              |</p>
<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td>Name of the dynamic response signal</td>
</tr>
<tr>
<td>Delay</td>
<td>Float</td>
<td>Delays the sending of the dynamic response signal</td>
</tr>
<tr>
<td>ContextCollection</td>
<td>String</td>
<td>Name of the variable collection sent along with the signal as a context</td>
</tr>
<tr>
<td>AutoReleaseContextCollection</td>
<td>Boolean</td>
<td>Controls whether the variable collection is released after processing the signal.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>String</td>
<td>Triggered when the signal is sent or is cancelled.</td>
</tr>
</tbody>
</table>

### SetFloatVariable node

Used to set a float variable in the Dynamic Response system.

```plaintext
Set a float variable in the Dynamic Response system.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>Any</td>
<td>Set the given value to the specified variable</td>
</tr>
<tr>
<td>EntityID</td>
<td>Any</td>
<td>The ID of the entity to fetch the collection from</td>
</tr>
<tr>
<td>CollectionName</td>
<td>String</td>
<td>The name of the collection</td>
</tr>
<tr>
<td>VariableName</td>
<td>String</td>
<td>The name of the variable to set</td>
</tr>
<tr>
<td>FloatValue</td>
<td>Float</td>
<td>The value of the variable</td>
</tr>
<tr>
<td>ResetTime</td>
<td>Float</td>
<td>The time after which the variable is reset to its previous value</td>
</tr>
<tr>
<td>GenerateNew</td>
<td>Boolean</td>
<td>Determines whether a new variable collection should be generated</td>
</tr>
</tbody>
</table>
```
Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UsedCollectionName</td>
<td>String</td>
<td>Outputs the name of the variable collection created or used</td>
</tr>
</tbody>
</table>

**SetIntegerVariable node**

Used to set a float variable in the Dynamic Response system.

```
set UsedCollectionName = EntityID, CollectionName= CollectionName= VariableName= FloatValue= ResetTime= GenerateNew= 0
```

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>Any</td>
<td>Set the given value to the specified variable</td>
</tr>
<tr>
<td>EntityID</td>
<td>Voif</td>
<td>The ID of the entity to fetch the collection from</td>
</tr>
<tr>
<td>CollectionName</td>
<td>String</td>
<td>The name of the collection</td>
</tr>
<tr>
<td>VariableName</td>
<td>String</td>
<td>The name of the variable to set</td>
</tr>
<tr>
<td>FloatValue</td>
<td>Float</td>
<td>The value of the variable</td>
</tr>
<tr>
<td>ResetTime</td>
<td>Float</td>
<td>The time after which the variable is reset to its previous value</td>
</tr>
<tr>
<td>GenerateNew</td>
<td>Boolean</td>
<td>Determines whether a new variable collection should be generated</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UsedCollectionName</td>
<td>String</td>
<td>Outputs the name of the variable collection created or used</td>
</tr>
</tbody>
</table>

**SetStringVariable node**

Used to set a string variable in the Dynamic Response system
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>Any</td>
<td>Set the given value to the specified variable</td>
</tr>
<tr>
<td>EntityID</td>
<td>Any</td>
<td>The ID of the entity to fetch the collection from</td>
</tr>
<tr>
<td>CollectionName</td>
<td>String</td>
<td>The name of the collection</td>
</tr>
<tr>
<td>VariableName</td>
<td>String</td>
<td>The name of the variable to set</td>
</tr>
<tr>
<td>FloatValue</td>
<td>String</td>
<td>The value of the variable</td>
</tr>
<tr>
<td>ResetTime</td>
<td>Float</td>
<td>The time after which the variable is reset to its previous value</td>
</tr>
<tr>
<td>GenerateNew</td>
<td>Boolean</td>
<td>Determines whether a new variable collection should be generated</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UsedCollectionName</td>
<td>String</td>
<td>Outputs the name of the variable collection created or used</td>
</tr>
</tbody>
</table>

Engine Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to configure various Lumberyard engine settings.

Topics

- LayerSwitch node (p. 314)
• PortalSwitch node (p. 314)
• PrecacheArea node (p. 314)
• Viewport node (p. 315)

LayerSwitch node

Used to activate and deactivate objects in a layer, as well as streaming in data to a layer.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer</td>
<td>String</td>
<td>Name of the layer</td>
</tr>
<tr>
<td>Hide</td>
<td>Any</td>
<td>Hides objects in the layer</td>
</tr>
<tr>
<td>Unhide</td>
<td>Any</td>
<td>Shows objects in the layer</td>
</tr>
<tr>
<td>EnableSerialization</td>
<td>Any</td>
<td>Enables objects in the layer</td>
</tr>
<tr>
<td>DisableSerialization</td>
<td>Any</td>
<td>Disables objects in the layer</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hidden</td>
<td>Any</td>
<td>Triggered if hidden</td>
</tr>
<tr>
<td>Unhidden</td>
<td>Any</td>
<td>Triggered if visible</td>
</tr>
</tbody>
</table>

PortalSwitch node

Used to switch the portal on or off.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the portal switch</td>
</tr>
<tr>
<td>Deactivate</td>
<td>Any</td>
<td>Deactivates the portal switch</td>
</tr>
</tbody>
</table>

PrecacheArea node

Used to precache an area at a specified location
### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>Vec3</td>
<td>Location of the area to be precached</td>
</tr>
<tr>
<td>Timeout</td>
<td>Float</td>
<td>Timeout interval in seconds</td>
</tr>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
</tbody>
</table>

### Viewport node

Used to get current viewport information.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get</td>
<td>Any</td>
<td>Gets the current viewport information</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Integer</td>
<td>Outputs the top left X position of the viewport</td>
</tr>
<tr>
<td>y</td>
<td>Integer</td>
<td>Outputs the top left Y position of the viewport</td>
</tr>
<tr>
<td>width</td>
<td>Integer</td>
<td>Outputs the width of the viewport</td>
</tr>
<tr>
<td>height</td>
<td>Integer</td>
<td>Outputs the height of the viewport</td>
</tr>
</tbody>
</table>

### Entity Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about **Script Canvas**, Lumberyard’s new visual scripting environment.
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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to control entity behavior and configure related settings.

**Topics**

- Attachment node (p. 316)
- BeamEntity node (p. 317)
- BroadcastEvent node (p. 318)
- CallScriptFunction node (p. 318)
- CharAttachmentMaterialParam node (p. 319)
- CheckDistance node (p. 319)
- ChildAttach node (p. 320)
- ChildDetach node (p. 321)
- Damage node (p. 321)
- EntitiesInRange node (p. 322)
- EntityId node (p. 322)
- EntityInfo node (p. 323)
- EntityPool node (p. 323)
- EntityPos node (p. 324)
- FindEntityByName node (p. 324)
- GetBounds node (p. 325)
- GetEntityExistence node (p. 325)
- GetPos node (p. 326)
- ParentId node (p. 326)
- PropertyGet node (p. 326)
- PropertySet node (p. 327)
- RemoveEntity node (p. 327)
- RenderParams node (p. 328)
- Spawn node (p. 328)
- SpawnArchetype node (p. 329)

**Attachment node**

Used to attach and detach attachments to an entity.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item</td>
<td>Any</td>
<td>Entity to be linked</td>
</tr>
<tr>
<td>BoneName</td>
<td>String</td>
<td>Attachment bone</td>
</tr>
<tr>
<td>CharacterSlot</td>
<td>Integer</td>
<td>Host character slot</td>
</tr>
<tr>
<td>Attach</td>
<td>Any</td>
<td>Attach entity attached to the bone</td>
</tr>
<tr>
<td>Detach</td>
<td>Any</td>
<td>Detach entity attached to bone</td>
</tr>
<tr>
<td>Hide</td>
<td>Any</td>
<td>Hide attachment</td>
</tr>
<tr>
<td>Unhide</td>
<td>Any</td>
<td>Show attachment</td>
</tr>
<tr>
<td>RotationOffset</td>
<td>Vec3</td>
<td>Rotation offset</td>
</tr>
<tr>
<td>TranslationOffset</td>
<td>Vec3</td>
<td>Translation offset</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attached</td>
<td>Any</td>
<td>Triggers when entity is attached</td>
</tr>
<tr>
<td>Detached</td>
<td>Any</td>
<td>Triggers when entity is detached</td>
</tr>
</tbody>
</table>

BeamEntity node

Used to beam or teleport objects instantly to any position in the level. When the Beam port is triggered the target entity is moved to the position input on the Position port.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beam</td>
<td>Any</td>
<td>Trigger to beam the entity</td>
</tr>
<tr>
<td>Position</td>
<td>Vec3</td>
<td>Destination location to beam to</td>
</tr>
<tr>
<td>Rotation</td>
<td>Vec3</td>
<td>Rotation to apply to entity</td>
</tr>
<tr>
<td>UseZeroRot</td>
<td>Boolean</td>
<td>Applies rotation even if it is 0</td>
</tr>
<tr>
<td>Scale</td>
<td>Vec3</td>
<td>Vector scale value</td>
</tr>
<tr>
<td>Memo</td>
<td>String</td>
<td>Memo to log when position is 0</td>
</tr>
</tbody>
</table>
## Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when entity has beamed to another location</td>
</tr>
</tbody>
</table>

### BroadcastEvent node

Used to send an event to one or more entities. The entities that will receive this event are specified by inputting a string to the **name** port. Each entity that has the string that is input there as a part of their name will receive the event set in the **event** port.

```plaintext
Entity:BroadcastEvent
# send
# event=
# name=
```

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>send</td>
<td>Any</td>
<td>Trigger to send an event</td>
</tr>
<tr>
<td>event</td>
<td>String</td>
<td>Event to be sent</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>Entity to receive the event</td>
</tr>
</tbody>
</table>

### CallScriptFunction node

Used to call a script function for the entity.

```plaintext
Entity:CallScriptFunction
# Choose Entity
# call
# FunctionName= 
# Argument1
# Argument2
# Argument3
# Argument4
# Argument5
```

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>call</td>
<td>Any</td>
<td>Calls the function</td>
</tr>
<tr>
<td>FunctionName</td>
<td>String</td>
<td>Script function name</td>
</tr>
<tr>
<td>Argument1 -</td>
<td>Any</td>
<td>Function arguments</td>
</tr>
<tr>
<td>Argument5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Script function was found and called</td>
</tr>
<tr>
<td>Failed</td>
<td>Any</td>
<td>Script function was not found or called</td>
</tr>
</tbody>
</table>
CharAttachmentMaterialParam node

Used to change a material on an attachment in a .cdf file. For example, you can change the material of a character's trousers.

Set Material is the trigger, ForcedMaterial is the full file path to the material (for example: materials/references/basecolors/grey.mtl) and SubMtlId is the number of the sub-material.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CharSlot</td>
<td>Integer</td>
<td>Character slot within the entity</td>
</tr>
<tr>
<td>Attachment</td>
<td>String</td>
<td>Attachment</td>
</tr>
<tr>
<td>SetMaterial</td>
<td>Any</td>
<td>Sets the material</td>
</tr>
<tr>
<td>ForcedMaterial</td>
<td>String</td>
<td>Forcefully set the material</td>
</tr>
<tr>
<td>SubMtlId</td>
<td>Integer</td>
<td>Submaterial ID</td>
</tr>
<tr>
<td>Get</td>
<td>Any</td>
<td>Trigger to get current value</td>
</tr>
<tr>
<td>ParamFloat</td>
<td>String</td>
<td>Float parameter to get or be set</td>
</tr>
<tr>
<td>ValueFloat</td>
<td>Float</td>
<td>Trigger to set value</td>
</tr>
<tr>
<td>ParamColor</td>
<td>String</td>
<td>Color parameter to get or be set</td>
</tr>
<tr>
<td>ValueColor</td>
<td>Vec3</td>
<td>Sets value color</td>
</tr>
</tbody>
</table>

CheckDistance node

Used to check the distance between the node entity and the entities defined in the input ports.
**Entity Nodes**

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check</td>
<td>Any</td>
<td>Trigger to check distance</td>
</tr>
<tr>
<td>MinDistance</td>
<td>Float</td>
<td>An entity that is nearer this distance will be ignored</td>
</tr>
<tr>
<td>MaxDistance</td>
<td>Float</td>
<td>An entity that is further than this distance will be ignored</td>
</tr>
<tr>
<td>Entity1 - Entity16</td>
<td>Any</td>
<td>Entity ID values</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NearEntity</td>
<td>Any</td>
<td>Nearest entity</td>
</tr>
<tr>
<td>NearEntityDist</td>
<td>Float</td>
<td>Distance of nearest entity</td>
</tr>
<tr>
<td>FarEntity</td>
<td>Any</td>
<td>Furthest entity</td>
</tr>
<tr>
<td>FarEntityDist</td>
<td>Float</td>
<td>Distance of furthest entity</td>
</tr>
<tr>
<td>NoEntInRange</td>
<td>Any</td>
<td>Triggers when no entities are between MinDistance and MaxDistance</td>
</tr>
</tbody>
</table>

**ChildAttach node**

Used to attach another entity to its target entity. The child entity will be linked to the target entity until the link is removed. The entity defined in the Child input port is attached to the target entity.
Lumberyard Legacy Reference
Entity Nodes

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attach</td>
<td>Any</td>
<td>Triggers entity attachment</td>
</tr>
<tr>
<td>Child</td>
<td>Any</td>
<td>Child entity to be attached</td>
</tr>
<tr>
<td>KeepTransform</td>
<td>Boolean</td>
<td>Child entity will be kept at the same transformation in world space</td>
</tr>
<tr>
<td>DisablePhysics</td>
<td>Boolean</td>
<td>Disable physics for child entity when attached</td>
</tr>
</tbody>
</table>

**ChildDetach node**

Used to detach entities from its parent entity. Usually the **ChildAttach** node has been used before to link the target entity to another entity.

When **KeepTransform** is set, the entity will keep its transformation in world space when detached. When **EnablePhysics** is set, physics will be re-enabled again when the entity is detached.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detach</td>
<td>Any</td>
<td>Triggers entity detachment</td>
</tr>
<tr>
<td>KeepTransform</td>
<td>Boolean</td>
<td>Child entity will be kept at the same transformation in world space</td>
</tr>
<tr>
<td>EnablePhysics</td>
<td>Boolean</td>
<td>Enable physics for child entity when detached</td>
</tr>
</tbody>
</table>

**Damage node**

Used to damage the specified entity when the trigger is activated.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>Damage</td>
<td>Integer</td>
<td>Amount of damage to inflict</td>
</tr>
<tr>
<td>DamageRelative</td>
<td>Integer</td>
<td>Damage inflicted is relative to the health of the entity</td>
</tr>
<tr>
<td>Position</td>
<td>Vec3</td>
<td>Location damage occurs at</td>
</tr>
</tbody>
</table>
**EntitiesInRange node**

Used to take the positions of two entities and check if they are in a certain range to each other. Depending on the result of the check the output ports are triggered.

![EntitiesInRange node diagram]

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>Entity1</td>
<td>Any</td>
<td>Entity 1</td>
</tr>
<tr>
<td>Entity2</td>
<td>Any</td>
<td>Entity 2</td>
</tr>
<tr>
<td>Range</td>
<td>Float</td>
<td>Distance range to check</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InRange</td>
<td>Boolean</td>
<td>True if entities are in range of each other</td>
</tr>
<tr>
<td>False</td>
<td>Any</td>
<td>Triggers if entities are not in range</td>
</tr>
<tr>
<td>True</td>
<td>Any</td>
<td>Triggers if entities are in range</td>
</tr>
<tr>
<td>Distance</td>
<td>Float</td>
<td>Floating point distance between the two entities</td>
</tr>
<tr>
<td>DistVec</td>
<td>Vec3</td>
<td>Vector distance between the two entities</td>
</tr>
</tbody>
</table>

**EntityId node**

Used to output the entity ID number of the specified entity. The node does not need to be triggered as the entity ID never changes.

![EntityId node diagram]

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Entity ID</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Any</td>
<td>Outputs the entity ID</td>
</tr>
</tbody>
</table>
EntityInfo node

Used to output the ID, name, class, and archetype of the target entity.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get</td>
<td>Any</td>
<td>Gets entity information</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Any</td>
<td>Entity ID</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>Entity name</td>
</tr>
<tr>
<td>Class</td>
<td>String</td>
<td>Entity class</td>
</tr>
<tr>
<td>Archetype</td>
<td>String</td>
<td>Entity archetype</td>
</tr>
</tbody>
</table>

EntityPool node

Used to prepare an entity from the pool or free it back to the pool.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prepare</td>
<td>Any</td>
<td>Brings entity into existence from the pool</td>
</tr>
<tr>
<td>Free</td>
<td>Any</td>
<td>Frees the entity back to the pool</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ready</td>
<td>Any</td>
<td>Triggers when the entity is prepared and ready</td>
</tr>
<tr>
<td>Freed</td>
<td>Any</td>
<td>Triggers when the entity is freed and returns to the pool</td>
</tr>
<tr>
<td>Error</td>
<td>Any</td>
<td>Triggers when an error occurs</td>
</tr>
</tbody>
</table>
EntityPos node

Handles all position related manipulations of the owner entity. All position information of the specified entity can be read from the output ports.

Unlike the GetPos node, the output ports of this node are triggered whenever one of the target entity properties changes.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>Vec3</td>
<td>Entity position</td>
</tr>
<tr>
<td>Rotate</td>
<td>Vec3</td>
<td>Entity rotation angle in degrees</td>
</tr>
<tr>
<td>Scale</td>
<td>Vec3</td>
<td>Entity scale</td>
</tr>
<tr>
<td>CoordSys</td>
<td>Integer</td>
<td>Coordinate system used</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>Vec3</td>
<td>Current entity position</td>
</tr>
<tr>
<td>Rotate</td>
<td>Vec3</td>
<td>Current entity rotation angle in degrees</td>
</tr>
<tr>
<td>Scale</td>
<td>Vec3</td>
<td>Current entity scale</td>
</tr>
<tr>
<td>FwdDir</td>
<td>Vec3</td>
<td>Current entity y-axis position</td>
</tr>
<tr>
<td>RightDir</td>
<td>Vec3</td>
<td>Current entity x-axis position</td>
</tr>
<tr>
<td>UpDir</td>
<td>Vec3</td>
<td>Current entity z-axis position</td>
</tr>
</tbody>
</table>

FindEntityByName node

Used to find an entity by name and output the entity ID.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>Any</td>
<td>Start searching for entity</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>Name of entity to look for</td>
</tr>
</tbody>
</table>
Lumberyard Legacy Reference
Entity Nodes

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntityId</td>
<td>Any</td>
<td>Outputs the entity ID if found</td>
</tr>
</tbody>
</table>

**GetBounds node**

Used to get and output the bounds.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get</td>
<td>Any</td>
<td>Gets the AABB bounding box</td>
</tr>
<tr>
<td>CoordSys</td>
<td>Integer</td>
<td>Coordinate system used</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>Vec3</td>
<td>Minimum position of the AABB</td>
</tr>
<tr>
<td>Max</td>
<td>Vec3</td>
<td>Maximum position of the AABB</td>
</tr>
</tbody>
</table>

**GetEntityExistence node**

Used to get an entity's existence.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get</td>
<td>Any</td>
<td>Gets entity existence status</td>
</tr>
<tr>
<td>EntityId</td>
<td>Any</td>
<td>Entity ID</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exists</td>
<td>Boolean</td>
<td>True if the entity exists</td>
</tr>
<tr>
<td>True</td>
<td>Any</td>
<td>Triggers if the entity exists</td>
</tr>
<tr>
<td>False</td>
<td>Any</td>
<td>Triggers if the entity exists</td>
</tr>
</tbody>
</table>
GetPos node

Used to output position information only when the trigger is activated. Similar to the EntityPos node, which triggers the output ports continuously whenever any position information changes.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get</td>
<td>Any</td>
<td>Gets entity position</td>
</tr>
<tr>
<td>CoordSys</td>
<td>Integer</td>
<td>Coordinate system used</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>Vec3</td>
<td>Entity position</td>
</tr>
<tr>
<td>Rotate</td>
<td>Vec3</td>
<td>Entity rotation</td>
</tr>
<tr>
<td>Scale</td>
<td>Vec3</td>
<td>Entity scale</td>
</tr>
<tr>
<td>FwdDir</td>
<td>Vec3</td>
<td>Entity y-axis position</td>
</tr>
<tr>
<td>RightDir</td>
<td>Vec3</td>
<td>Entity x-axis position</td>
</tr>
<tr>
<td>UpDir</td>
<td>Vec3</td>
<td>Entity z-axis position</td>
</tr>
</tbody>
</table>

ParentId node

Used to obtain the parentID number of the specified entity.

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent Id</td>
<td>Any</td>
<td>Parent entity ID</td>
</tr>
</tbody>
</table>

PropertyGet node

Used to retrieve an entity property value.
Lumberyard Legacy Reference
Entity Nodes

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get</td>
<td>Any</td>
<td>Trigger to get entity property value</td>
</tr>
<tr>
<td>Property</td>
<td>String</td>
<td>Name of property to get</td>
</tr>
<tr>
<td>PerArchetype</td>
<td>Boolean</td>
<td>True if a per archetype property; false if a per instance property</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Any</td>
<td>Outputs property value</td>
</tr>
<tr>
<td>Error</td>
<td>Any</td>
<td>Retrieves property value</td>
</tr>
</tbody>
</table>

**PropertySet node**

Used to change the entity property value. Will not work with SaveLoad however.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>Any</td>
<td>Sets property value</td>
</tr>
<tr>
<td>Property</td>
<td>String</td>
<td>Name of property to set</td>
</tr>
<tr>
<td>Value</td>
<td>String</td>
<td>Property value to be set</td>
</tr>
<tr>
<td>PerArchetype</td>
<td>Boolean</td>
<td>True if a per archetype property; false if a per instance property</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>Any</td>
<td>Any</td>
</tr>
</tbody>
</table>

**RemoveEntity node**

Used to remove an entity.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>Entity</td>
<td>Integer</td>
<td>Entity to remove</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when entity has been removed</td>
</tr>
</tbody>
</table>

**RenderParams node**

Used to set rendering parameters.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opacity</td>
<td>Float</td>
<td>Sets entity transparency value</td>
</tr>
</tbody>
</table>

**Spawn node**

Used to spawn an entity with the specified properties.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spawn</td>
<td>Any</td>
<td>Spawns an entity</td>
</tr>
<tr>
<td>Class</td>
<td>String</td>
<td>Entity class</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>Entity class</td>
</tr>
<tr>
<td>Pos</td>
<td>Vec3</td>
<td>Entity position</td>
</tr>
<tr>
<td>Rot</td>
<td>Vec3</td>
<td>Entity rotation</td>
</tr>
<tr>
<td>Scale</td>
<td>Vec3</td>
<td>Entity scale</td>
</tr>
</tbody>
</table>
### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when entity has completed spawning</td>
</tr>
<tr>
<td>Succeeded</td>
<td>Any</td>
<td>Triggers when entity is spawned</td>
</tr>
<tr>
<td>Failed</td>
<td>Any</td>
<td>Triggers if entity was not spawned</td>
</tr>
</tbody>
</table>

### SpawnArchetype node

Used to spawn an archetype entity with the specified properties.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spawn</td>
<td>Any</td>
<td>Spawns an entity</td>
</tr>
<tr>
<td>Archetype</td>
<td>String</td>
<td>Archetype entity name</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>Entity name</td>
</tr>
<tr>
<td>Pos</td>
<td>Vec3</td>
<td>Entity position</td>
</tr>
<tr>
<td>Rot</td>
<td>Vec3</td>
<td>Entity rotation angle</td>
</tr>
<tr>
<td>Scale</td>
<td>Vec3</td>
<td>Entity scale</td>
</tr>
</tbody>
</table>

### Environment Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.
You can use the following flow graph nodes to configure environment settings.

**Topics**

- MoonDirection node (p. 330)
- OceanSwitch node (p. 330)
- PerEntityShadows node (p. 331)
- RainProperties node (p. 331)
- RecomputeStaticShadows node (p. 332)
- SetOceanMaterial node (p. 332)
- SkyMaterialSwitch node (p. 332)
- SkyboxSwitch node (p. 333)
- Sun node (p. 333)
- TornadoWander (p. 334)
- Wind node (p. 334)

### MoonDirection node

Used to set the moon's position in the sky.

![MoonDirection Node](image)

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get</td>
<td>Any</td>
<td>Get current latitude and longitude</td>
</tr>
<tr>
<td>Set</td>
<td>Any</td>
<td>Set latitude and longitude</td>
</tr>
<tr>
<td>Latitude</td>
<td>Float</td>
<td>Latitude to be set</td>
</tr>
<tr>
<td>Longitude</td>
<td>Float</td>
<td>Longitude to be set</td>
</tr>
<tr>
<td>ForceUpdate</td>
<td>Boolean</td>
<td>Force immediate update of the sky</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latitude</td>
<td>Float</td>
<td>Output current latitude</td>
</tr>
<tr>
<td>Longitude</td>
<td>Float</td>
<td>Output current longitude</td>
</tr>
</tbody>
</table>

### OceanSwitch node

Used to enable ocean rendering.

![OceanSwitch Node](image)
Lumberyard Legacy Reference
Environment Nodes

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Enable ocean rendering</td>
</tr>
</tbody>
</table>

PerEntityShadows node

Used to enable and specify per entity shadows.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Boolean</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Triggers the parameters</td>
</tr>
<tr>
<td>ConstBias</td>
<td>Float</td>
<td>Reduces any self-shadowing artifacts</td>
</tr>
<tr>
<td>SlopeBias</td>
<td>Float</td>
<td>Reduces any self-shadowing artifacts</td>
</tr>
<tr>
<td>Jittering</td>
<td>Float</td>
<td>Filters kernel size, which directly affects shadow softness</td>
</tr>
<tr>
<td>BBoxScale</td>
<td>Vec3</td>
<td>Scale factor for the bounding box of the selected entity. Can be useful in case the bounding box is too small or too large</td>
</tr>
<tr>
<td>ShadowMapSize</td>
<td>Integer</td>
<td>Size of the custom shadow map, which is automatically rounded to the next power of two</td>
</tr>
</tbody>
</table>

RainProperties node

Used to get and output rain properties.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Amount</td>
<td>Float</td>
<td>Amount of rain</td>
</tr>
<tr>
<td>PuddlesAmount</td>
<td>Float</td>
<td>Amount of puddles</td>
</tr>
</tbody>
</table>
### PuddlesRippleAmount Port
- **Type**: Float
- **Description**: Amount of puddle ripples

### RainDropsAmount Port
- **Type**: Float
- **Description**: Amount of raindrops

## RecomputeStaticShadows node
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. Use this node to override this automated placement.

### Input

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Min</td>
<td>Vec3</td>
<td>Minimum bounding box position</td>
</tr>
<tr>
<td>Max</td>
<td>Vec3</td>
<td>Maximum bounding box position</td>
</tr>
<tr>
<td>NextCascadesScale</td>
<td>Float</td>
<td>Input multiplier value</td>
</tr>
</tbody>
</table>

## SetOceanMaterial node
Used to set the ocean material.

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>Any</td>
<td>Set material on for the ocean</td>
</tr>
<tr>
<td>Material</td>
<td>String</td>
<td>Material to be set for the ocean</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Triggered when material set</td>
</tr>
<tr>
<td>Failed</td>
<td>Any</td>
<td>Triggered if an error occurred</td>
</tr>
</tbody>
</table>

## SkyMaterialSwitch node
Used to enable sky material switching.
Lumberyard Legacy Reference
Environment Nodes

**SkyboxSwitch node**

Used to enable asynchronous sky box switching.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skybox texture name</td>
<td>String</td>
<td>Name of texture file to use</td>
</tr>
<tr>
<td>Start</td>
<td>Boolean</td>
<td>Start asynchronous switching</td>
</tr>
<tr>
<td>Angle</td>
<td>Float</td>
<td>Starting angle</td>
</tr>
<tr>
<td>Stretching</td>
<td>Float</td>
<td>If stretching is performed or not</td>
</tr>
</tbody>
</table>

**Sun node**

Used to get and set the sun's position in the sky.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get</td>
<td>Any</td>
<td>Get the current latitude and longitude</td>
</tr>
<tr>
<td>Set</td>
<td>Any</td>
<td>Set the latitude and longitude for the sun</td>
</tr>
<tr>
<td>Latitude</td>
<td>Float</td>
<td>Latitude to be set</td>
</tr>
<tr>
<td>Longitude</td>
<td>Float</td>
<td>Longitude to be set</td>
</tr>
</tbody>
</table>

**Environment Nodes**

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Material</td>
<td>String</td>
<td>Material to use for the sky</td>
</tr>
<tr>
<td>Start</td>
<td>Boolean</td>
<td>Start material switch</td>
</tr>
<tr>
<td>Angle</td>
<td>Float</td>
<td>Starting angle</td>
</tr>
<tr>
<td>Stretching</td>
<td>Float</td>
<td>If stretching is performed or not</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skybox texture name</td>
<td>String</td>
<td>Name of texture file to use</td>
</tr>
<tr>
<td>Start</td>
<td>Boolean</td>
<td>Start asynchronous switching</td>
</tr>
<tr>
<td>Angle</td>
<td>Float</td>
<td>Starting angle</td>
</tr>
<tr>
<td>Stretching</td>
<td>Float</td>
<td>If stretching is performed or not</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get</td>
<td>Any</td>
<td>Get the current latitude and longitude</td>
</tr>
<tr>
<td>Set</td>
<td>Any</td>
<td>Set the latitude and longitude for the sun</td>
</tr>
<tr>
<td>Latitude</td>
<td>Float</td>
<td>Latitude to be set</td>
</tr>
<tr>
<td>Longitude</td>
<td>Float</td>
<td>Longitude to be set</td>
</tr>
<tr>
<td>Port</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>ForceUpdate</td>
<td>Boolean</td>
<td>Forces an immediate update of the sky</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latitude</td>
<td>Float</td>
<td>Outputs current latitude</td>
</tr>
<tr>
<td>Longitude</td>
<td>Float</td>
<td>Outputs current longitude</td>
</tr>
</tbody>
</table>

### TornadoWander

Used to move a tornado entity in the direction of the target.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Target</td>
<td>Any</td>
<td>Location the tornado moves towards</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when the tornado reaches the target</td>
</tr>
</tbody>
</table>

### Wind node

Used to get and output the wind direction vector.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get</td>
<td>Any</td>
<td>Get the current environment wind vector</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WindVector</td>
<td>Vec3</td>
<td>Outputs current environment wind vector</td>
</tr>
</tbody>
</table>
FeatureTest Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard’s new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to configure feature test settings.

Topics
- FeatureTest node (p. 335)
- Screenshot node (p. 336)
- ScreenshotCompare node (p. 336)

FeatureTest node

Used to control automated feature tests.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td>Name of the feature test</td>
</tr>
<tr>
<td>Description</td>
<td>String</td>
<td>Description of the feature test</td>
</tr>
<tr>
<td>MaxTime</td>
<td>Float</td>
<td>How long in game time the test is allowed to run before it fails</td>
</tr>
<tr>
<td>Camera</td>
<td>Any</td>
<td>(Optional) Camera entity used for the test</td>
</tr>
<tr>
<td>Ready</td>
<td>Boolean</td>
<td>Indicates whether all dependencies have been met and the test is ready to run</td>
</tr>
<tr>
<td>Succeeded</td>
<td>Any</td>
<td>Trigger to indicate the feature test has passed</td>
</tr>
<tr>
<td>Failed</td>
<td>Any</td>
<td>Trigger to indicate the feature test has failed</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Trigger to start running the feature test</td>
</tr>
</tbody>
</table>
Screenshot node

Used to take a screenshot.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Trigger to capture a screenshot</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>Name of the screenshot</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Succeeded</td>
<td>Any</td>
<td>Triggers when the image is captured</td>
</tr>
<tr>
<td>Failed</td>
<td>Any</td>
<td>Triggers if the image is not captured</td>
</tr>
</tbody>
</table>

ScreenshotCompare node

Used to take a screenshot and compare it with a reference image.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Trigger to capture a screenshot</td>
</tr>
<tr>
<td>Reset</td>
<td>Any</td>
<td>Resets the current screenshot number back to 0</td>
</tr>
<tr>
<td>PSNR</td>
<td>Float</td>
<td>Picture signal to noise ratio used during comparison with the reference image to determine success of failure</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Succeeded</td>
<td>Any</td>
<td>Triggers when the captured image matches the reference image</td>
</tr>
<tr>
<td>Failed</td>
<td>Any</td>
<td>Triggers when the captured image does not match the reference image</td>
</tr>
</tbody>
</table>
Force Feedback Nodes

Lumberyard uses the force feedback system to manage haptic feedback for motion controllers. You can use the force feedback nodes to configure haptic feedback settings.

Topics
- Game:ForceFeedbackSetDeviceIndex (p. 337)
- Game:ForceFeedback (p. 337)
- Game:ForceFeedbackTriggerTweaker (p. 339)
- Game:ForceFeedbackTweaker (p. 340)

Game:ForceFeedbackSetDeviceIndex

Sets the device ID for force feedback effects.

Node Inputs

DeviceIndex

Enables or disables the force feedback functionality.

Valid values: 0 = Enable | -1 = Disable

Game:ForceFeedback

Controls the force feedback effect with an effect that is specified in the ForceFeedbackEffects.xml file (located in the \libs\GameForceFeedback directory).

The following is an example ForceFeedbackEffects.xml file with right_shoot and left_shoot effects added:

```xml
<ForceFeedback>
  <Patterns>
    <!-- Pattern definition accepts from 1 to 16 samples -->
    <Pattern name="100" samples="1" />  
    <Pattern name="75" samples="0.75" />
    <Pattern name="50" samples="0.5" />  
    <Pattern name="30" samples="0.3" />  
    <Pattern name="20" samples="0.2" />  
    <Pattern name="12" samples="0.12" />
  </Patterns>
  <Envelopes>
    <!-- Envelope definition accepts from 1 to 8 samples -->
    <Envelope name="linearFalloff" samples="1,0" />
  </Envelopes>
</ForceFeedback>
```
Node Inputs

**Effect Name**
Select the effect to use from the Effect Name drop-down list.

**Play**
Plays the force feedback effect.

**Intensity**
Applies the intensity factor to the effect being played.
Default value: 1

**Delay**
Time delay to start the effect.
Default value: 0

**Stop**
Stops the force feedback effect.
StopAll

Stops all force feedback effects.

Input Device Type

Specifies the input device to which to send force feedback.

Default value: Gamepad

Valid values: Gamepad | Joystick | Keyboard | MotionController | MotionSensor | Mouse | TouchScreen

Note
MotionController uses Left, LowPass, and MotorA terminology for the Left Controller. It uses Right, HighPass, and MotorB terminology for the Right Controller.

Game:ForceFeedbackTriggerTweaker

Controls the force feedback effect on the left and right triggers.

Node Inputs

LeftTouchToActivate

The left trigger's gain is modulated by how much the trigger is pressed.

Default value: 1

LeftGain

Gain sent to the left trigger's motor.

Default value: 1

LeftEnvelope

Envelope sent to the left trigger's motor.

Default value: 1

Valid values: 0 – 2000 (in multiples of 4)

RightTouchToActivate

The right trigger's gain is modulated by how much the trigger is pressed.

Default value: 1

RightGain

Gain sent to the right trigger's motor.

Default value: 1

RightEnvelope

Envelope sent to the right trigger's motor.

Default value: 1

Valid values: 0 – 2000 (in multiples of 4)

Activate

Activates the effect on the left and right triggers.
**Deactivate**

Deactivates the effect on the left and right triggers.

**Input Device Type**

Specifies the input device to which to send force feedback.

Default value: Gamepad

Valid values: Gamepad | Joystick | Keyboard | MotionController | MotionSensor | Mouse | TouchScreen

**Note**

MotionController uses Left, LowPass, and MotorA terminology for the Left Controller. It uses Right, HighPass, and MotorB terminology for the Right Controller.

**Game:ForceFeedbackTweaker**

Controls individual high or low frequency force feedback effects.

**Node Inputs**

**LowPass**

Applies the multiplier to the low frequency effect being played.

Default value: 1

**HighPass**

Applies the multiplier to the high frequency effect being played.

Default value: 1

**Activate**

Activates the high or low frequency force feedback effect.

**Deactivate**

Deactivates the high or low frequency force feedback effect.

**Input Device Type**

Specifies the input device to which to send force feedback.

Default value: Gamepad

Valid values: Gamepad | Joystick | Keyboard | MotionController | MotionSensor | Mouse | TouchScreen

**Note**

MotionController uses Left, LowPass, and MotorA terminology for the Left Controller. It uses Right, HighPass, and MotorB terminology for the Right Controller.

**Game Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.
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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to check and to configure various game settings.

**Topics**
- CheckPlatform node (p. 341)
- ForceFeedback node (p. 341)
- ForceFeedbackSetDeviceIndex node (p. 342)
- ForceFeedbackTriggerTweaker node (p. 342)
- ForceFeedbackTweaker node (p. 343)
- GetClientActorId node (p. 343)
- GetEntityState node (p. 343)
- GetGameRulesEntityId node (p. 344)
- GetSupportedGameRulesForMap node (p. 344)
- GetUsername node (p. 345)
- IsLevelOfType node (p. 345)
- ObjectEvent node (p. 345)
- Start node (p. 346)

**CheckPlatform node**

Used to change game events depending on what platform you are running on.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check</td>
<td>Any</td>
<td>Triggers a check of the current platform</td>
</tr>
</tbody>
</table>

**Inputs**

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>Any</td>
<td>Triggers if the platform is PC</td>
</tr>
<tr>
<td>Android</td>
<td>Any</td>
<td>Triggers if the platform is Android</td>
</tr>
<tr>
<td>iOS</td>
<td>Any</td>
<td>Triggers if the platform is iOS</td>
</tr>
</tbody>
</table>

**ForceFeedback node**

Used to start and stop force feedback effects.
## Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Effect Name</td>
<td>String</td>
<td>Name of the force feedback effect</td>
</tr>
<tr>
<td>Play</td>
<td>Any</td>
<td>Plays the effect</td>
</tr>
<tr>
<td>Intensity</td>
<td>Float</td>
<td>Intensity level of effect</td>
</tr>
<tr>
<td>Delay</td>
<td>Float</td>
<td>Delays effect start by specified seconds</td>
</tr>
<tr>
<td>Stop</td>
<td>Any</td>
<td>Stops the effect</td>
</tr>
<tr>
<td>StopAll</td>
<td>Any</td>
<td>Stops all effects</td>
</tr>
</tbody>
</table>

### ForceFeedbackSetDeviceIndex node

Used to set the receiving device ID for force feedback effects.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeviceIndex</td>
<td>Integer</td>
<td>Sets the receiving device ID for force feedback effects</td>
</tr>
</tbody>
</table>

### ForceFeedbackTriggerTweaker node

Used to control individual left and right trigger force feedback effects.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeftTouchActivate</td>
<td>Boolean</td>
<td>Activates the left touch trigger</td>
</tr>
<tr>
<td>LeftGain</td>
<td>Float</td>
<td>Left trigger gain</td>
</tr>
<tr>
<td>LeftEnvelope</td>
<td>Integer</td>
<td>Left trigger envelope</td>
</tr>
</tbody>
</table>
### ForceFeedbackTweaker node

Used to control individual low and high frequency force feedback effects.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LowPass</td>
<td>Float</td>
<td>Low-frequency force feedback signal</td>
</tr>
<tr>
<td>HighPass</td>
<td>Float</td>
<td>High-frequency force feedback signal</td>
</tr>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates force feedback effect</td>
</tr>
<tr>
<td>Deactivate</td>
<td>Any</td>
<td>Deactivates force feedback effect</td>
</tr>
</tbody>
</table>

### GetClientActorId node

Used to output the client actor ID.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Any</td>
<td>Gets client actor ID</td>
</tr>
</tbody>
</table>

### GetEntityState node

Used to output the current state of an entity.
Game Nodes

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get</td>
<td>Any</td>
<td>Gets the entity state</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>String</td>
<td>Outputs the entity state</td>
</tr>
</tbody>
</table>

GetGameRulesEntityId node

Used to get the game rules entity ID.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get</td>
<td>Any</td>
<td>Gets the entity ID of the rules script</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntityId</td>
<td>Any</td>
<td>The entity ID of the rules script</td>
</tr>
</tbody>
</table>

GetSupportedGameRulesForMap node

Used to get and output the supported game rules for a map.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get</td>
<td>Any</td>
<td>Gets the game rules</td>
</tr>
<tr>
<td>Mapname</td>
<td>String</td>
<td>Map name</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GameRules</td>
<td>String</td>
<td>Outputs the game rules</td>
</tr>
</tbody>
</table>
GetUsername node

Used to get the user name.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetUsername</td>
<td>Any</td>
<td>Gets the user name</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Username</td>
<td>String</td>
<td>Outputs the user name</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Triggers if an error occurs</td>
</tr>
</tbody>
</table>

IsLevelOfType node

Used to check if a level is of a given type.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check</td>
<td>Any</td>
<td>Checks if a level is of a given type</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
<td>Level type to check against</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Boolean</td>
<td>The result of the check</td>
</tr>
</tbody>
</table>

ObjectEvent node

Used to broadcast a game object event or send it to a specific entity.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>EventName</td>
<td>String</td>
<td>Game object event name</td>
</tr>
<tr>
<td>EventParam</td>
<td>String</td>
<td>Game object event parameter</td>
</tr>
</tbody>
</table>

**Start node**

Fires on the start of a game and used to trigger flow graphs upon level start.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InGame</td>
<td>Boolean</td>
<td>Triggers game mode to start</td>
</tr>
<tr>
<td>InEditor</td>
<td>Boolean</td>
<td>Triggers editor game mode to start</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>output</td>
<td>Boolean</td>
<td>Outputs the game mode</td>
</tr>
</tbody>
</table>

**Helicopter Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to configure flying vehicle and flight AI-related settings.

**Note**

These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at Lumberyard Downloads.

**Topics**

- EnableCombatMode node (p. 347)
- EnableFiring node (p. 347)
- FollowPath node (p. 347)
- ForceFire node (p. 348)
EnableCombatMode node

Used to alter the path the flight AI should follow so as to find the best position from which to engage the target.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables combat mode</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables combat mode</td>
</tr>
</tbody>
</table>

EnableFiring node

Used to enable the flight AI to fire at a target when used in combination with the EnableCombatMode node.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables firing mode</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables firing mode</td>
</tr>
</tbody>
</table>

FollowPath node

Used to set the path that the flight AI should follow.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Start following the path</td>
</tr>
<tr>
<td>Stop</td>
<td>Any</td>
<td>Stop following the path</td>
</tr>
<tr>
<td>PathName</td>
<td>String</td>
<td>Name of the path to follow</td>
</tr>
<tr>
<td>LoopPath</td>
<td>Boolean</td>
<td>How many times to loop around the path</td>
</tr>
<tr>
<td>Speed</td>
<td>Float</td>
<td>Speed of the flight AI</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrivedAtEnd</td>
<td>Any</td>
<td>Triggers when flight AI is at the end of the path</td>
</tr>
<tr>
<td>ArrivedNearToEnd</td>
<td>Any</td>
<td>Triggers when flight AI is near the end of the path</td>
</tr>
<tr>
<td>Stopped</td>
<td>Any</td>
<td>Triggers when flight AI has stopped</td>
</tr>
</tbody>
</table>

ForceFire node

Used to force the attention target of the flight AI to a specific entity.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables force firing</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables force firing</td>
</tr>
<tr>
<td>Target</td>
<td>Any</td>
<td>Attention target</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finished</td>
<td>Any</td>
<td>Triggers when finished</td>
</tr>
</tbody>
</table>

Image Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to configure various visual effects and image settings.

Topics
- 3DHudInterference node (p. 349)
- ColorCorrection node (p. 350)
- EffectAlienInterference node (p. 351)
- EffectBloodSplats node (p. 351)
• EffectDepthOfField node (p. 351)
• EffectFrost node (p. 352)
• EffectGhosting node (p. 352)
• EffectGroup node (p. 353)
• EffectRainDrops node (p. 353)
• EffectVolumetricScattering node (p. 354)
• EffectWaterDroplets node (p. 354)
• EffectWaterFlow node (p. 355)
• FilterBlur node (p. 355)
• FilterChromaShift node (p. 355)
• FilterDirectionalBlur node (p. 356)
• FilterGrain node (p. 356)
• FilterRadialBlur node (p. 356)
• FilterSharpen node (p. 357)
• FilterVisualArtifacts node (p. 357)
• ScreenCapture node (p. 358)
• ScreenFader node (p. 359)
• SetShadowMode node (p. 359)

3DHudInterference node

Used to add distortion effects to the HUD.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Enables the effect</td>
</tr>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Disables the effect</td>
</tr>
<tr>
<td>Amount</td>
<td>Float</td>
<td>Interference amount</td>
</tr>
<tr>
<td>Disrupt scale</td>
<td>Float</td>
<td>Disruption scale</td>
</tr>
<tr>
<td>Disrupt movement scale</td>
<td>Float</td>
<td>Disruption movement scale</td>
</tr>
<tr>
<td>Random grain strength scale</td>
<td>Float</td>
<td>Random grain strength scale</td>
</tr>
</tbody>
</table>
### ColorCorrection node

Used to control basic image settings such as saturation, contrast, brightness, and color.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Enables color correction</td>
</tr>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Disables color correction</td>
</tr>
<tr>
<td>Cyan</td>
<td>Any</td>
<td>Cyan increase or decrease</td>
</tr>
<tr>
<td>Magenta</td>
<td>Any</td>
<td>Magenta increase or decrease</td>
</tr>
<tr>
<td>Yellow</td>
<td>Any</td>
<td>Yellow increase or decrease</td>
</tr>
<tr>
<td>Luminance</td>
<td>Any</td>
<td>Luminance increase or decrease</td>
</tr>
<tr>
<td>Brightness</td>
<td>Any</td>
<td>Brightness increase or decrease</td>
</tr>
<tr>
<td>Contrast</td>
<td>Any</td>
<td>Contrast increase or decrease</td>
</tr>
<tr>
<td>Saturation</td>
<td>Any</td>
<td>Saturation increase or decrease</td>
</tr>
<tr>
<td>Hue</td>
<td>Any</td>
<td>Hue increase or decrease</td>
</tr>
</tbody>
</table>
**EffectAlienInterference node**

Used to add distortion effects to the players view, but doesn't affect the HUD.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Enables the effect</td>
</tr>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Disables the effect</td>
</tr>
<tr>
<td>Amount</td>
<td>Float</td>
<td>Intensity level of the effect</td>
</tr>
<tr>
<td>Color</td>
<td>Vec3</td>
<td>Color of the effect</td>
</tr>
</tbody>
</table>

**EffectBloodSplats node**

Used to place blood splats on the screen when used. Type=0 is human and Type =1 is alien. The Spawn input generates new blood splats.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Enables the effect</td>
</tr>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Disables the effect</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
<td>Type of effect</td>
</tr>
<tr>
<td>Amount</td>
<td>Float</td>
<td>Intensity level of the effect</td>
</tr>
<tr>
<td>Spawn</td>
<td>Boolean</td>
<td>Where the effect spawns at</td>
</tr>
</tbody>
</table>

**EffectDepthOfField node**

Used to add a depth of field effect, giving control over distance, range, and amount.
Lumberyard Legacy Reference
Image Nodes

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Disables the node</td>
</tr>
<tr>
<td>EnableDof</td>
<td>Boolean</td>
<td>Enables the depth of field effect</td>
</tr>
<tr>
<td>FocusDistance</td>
<td>Float</td>
<td>Sets the focus distance</td>
</tr>
<tr>
<td>FocusRange</td>
<td>Float</td>
<td>Sets the focus range</td>
</tr>
<tr>
<td>BlurAmount</td>
<td>Float</td>
<td>Sets the amount of blurring</td>
</tr>
<tr>
<td>ScaleCoC</td>
<td>Float</td>
<td>Sets the circle of confusion scale, which is the optical spot caused by a light rays cone from a lens not coming to a perfect focus when imaging a point source. Also known as the blur circle of blur spot.</td>
</tr>
<tr>
<td>CenterWeight</td>
<td>Float</td>
<td>Sets the central samples weight</td>
</tr>
</tbody>
</table>

**EffectFrost node**

Used to simulate a frozen HUD.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Enables the effect</td>
</tr>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Disables the effect</td>
</tr>
<tr>
<td>Amount</td>
<td>Any</td>
<td>Intensity level of the effect</td>
</tr>
<tr>
<td>CenterAmount</td>
<td>Any</td>
<td>Center of the effect</td>
</tr>
</tbody>
</table>

**EffectGhosting node**

Used to add a ghosting effect to the screen that overlaps and blurs previous frames together.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Enables the effect</td>
</tr>
</tbody>
</table>
### Image Nodes

#### EffectGroup node

Used to enable the specified effect group.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Any</td>
<td>Enables the effect group</td>
</tr>
<tr>
<td>Disabled</td>
<td>Any</td>
<td>Disables the effect group</td>
</tr>
<tr>
<td>GroupName</td>
<td>String</td>
<td>Name of effect group</td>
</tr>
</tbody>
</table>

#### EffectRainDrops node

Used to add on-screen rain drops that travel down the player's HUD.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Disables the node</td>
</tr>
<tr>
<td>Amount</td>
<td>Float</td>
<td>Sets raindrop visibility</td>
</tr>
<tr>
<td>Spawn Time Distance</td>
<td>Float</td>
<td>Sets raindrop spawn time distance</td>
</tr>
<tr>
<td>Size</td>
<td>Float</td>
<td>Size of rain drops</td>
</tr>
<tr>
<td>Size Variation</td>
<td>Float</td>
<td>Amount of variation in size of rain drops</td>
</tr>
<tr>
<td>Moisture Amount</td>
<td>Float</td>
<td>Sets moisture visibility area size</td>
</tr>
<tr>
<td>Moisture Hardness</td>
<td>Float</td>
<td>Sets noise texture blending factor</td>
</tr>
</tbody>
</table>
### Image Nodes

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moisture Droplet Amount</td>
<td>Float</td>
<td>Sets droplet texture blending factor</td>
</tr>
<tr>
<td>Moisture Variation</td>
<td>Float</td>
<td>Sets moisture variation</td>
</tr>
<tr>
<td>Moisture Speed</td>
<td>Float</td>
<td>Sets moisture animation speed</td>
</tr>
<tr>
<td>Moisture Fog Amount</td>
<td>Float</td>
<td>Sets amount of fog in moisture</td>
</tr>
</tbody>
</table>

#### EffectVolumetricScattering node

Used to add a volumetric effect useful for simulating snowy environments. With the ability to control color, speed, and amount, you can simulate various environments, such as lava.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Disables the node</td>
</tr>
<tr>
<td>Amount</td>
<td>Float</td>
<td>Sets the amount of volumetric scattering</td>
</tr>
<tr>
<td>Tiling</td>
<td>Float</td>
<td>Sets the volumetric scattering tiling</td>
</tr>
<tr>
<td>Speed</td>
<td>Float</td>
<td>Sets the volumetric scattering animation speed</td>
</tr>
<tr>
<td>Color</td>
<td>Vec3</td>
<td>Sets the volumetric scattering color</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
<td>Defines the type of volumetric scattering</td>
</tr>
</tbody>
</table>

#### EffectWaterDroplets node

Used to add a water effect that appears from various sources on the screen. Unlike the RainDroplets node, this simulates more of a splash-type effect of water being thrown on the screen in various places.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Enables the effect</td>
</tr>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Disables the effect</td>
</tr>
</tbody>
</table>
### EffectWaterFlow node

*Used to simulate dense water running down the screen, such as standing under a waterfall.*

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Enables the effect</td>
</tr>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Disables the effect</td>
</tr>
<tr>
<td>Amount</td>
<td>Float</td>
<td>Intensity level of effect</td>
</tr>
</tbody>
</table>

---

### FilterBlur node

*Used to Gaussian blur the entire screen, useful for simulating dense smoke affecting the player's eyes.*

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Enables the filter</td>
</tr>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Disables the filter</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
<td>Type of effect</td>
</tr>
<tr>
<td>Amount</td>
<td>Float</td>
<td>Intensity level of effect</td>
</tr>
</tbody>
</table>

---

### FilterChromaShift node

*Used to shift the chrominance information of the image. Best used in small amounts to create subtle film effects.*

---

355
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Enables the filter</td>
</tr>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Disables the filter</td>
</tr>
<tr>
<td>Amount</td>
<td>Float</td>
<td>Intensity level of filter</td>
</tr>
</tbody>
</table>

**FilterDirectionalBlur node**

Used to apply a blur in a specified direction based on movement.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Enables the filter</td>
</tr>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Disables the filter</td>
</tr>
<tr>
<td>Direction</td>
<td>Vec3</td>
<td>Direction of blurring effect</td>
</tr>
</tbody>
</table>

**FilterGrain node**

Used to set a grain filter.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Enables the filter</td>
</tr>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Disables the filter</td>
</tr>
<tr>
<td>Amount</td>
<td>Float</td>
<td>Intensity level of filter</td>
</tr>
</tbody>
</table>

**FilterRadialBlur node**

Used to blur the screen around a defined 2D position on the screen.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Enables the filter</td>
</tr>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Disables the filter</td>
</tr>
<tr>
<td>Amount</td>
<td>Float</td>
<td>Intensity level of filter</td>
</tr>
<tr>
<td>ScreenPosX</td>
<td>Float</td>
<td>X-axis center of blurring effect</td>
</tr>
<tr>
<td>ScreenPosY</td>
<td>Float</td>
<td>Y-axis center of blurring effect</td>
</tr>
<tr>
<td>BlurringRadius</td>
<td>Float</td>
<td>Radius of blurring effect</td>
</tr>
</tbody>
</table>

FilterSharpen node

Used to add sharpening to the image. You can use negative values to blur the screen also.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Enables the filter</td>
</tr>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Disables the filter</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
<td>Type of filter</td>
</tr>
<tr>
<td>Amount</td>
<td>Float</td>
<td>Intensity level of filter</td>
</tr>
</tbody>
</table>

FilterVisualArtifacts node

Used to apply numerous effects typically associate with old television sets, such as grain, vsync, interlacing, and pixelation. You can mask the effect using a texture, or apply it to the whole screen.
**Image Nodes**

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Boolean</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Boolean</td>
<td>Disables the node</td>
</tr>
<tr>
<td>VSync</td>
<td>Float</td>
<td>Amount of visible vsync</td>
</tr>
<tr>
<td>VSync frequency</td>
<td>Float</td>
<td>Vsync frequency</td>
</tr>
<tr>
<td>Interlacing</td>
<td>Float</td>
<td>Amount of visible interlacing</td>
</tr>
<tr>
<td>Interlacing tiling</td>
<td>Float</td>
<td>Interlacing tiling</td>
</tr>
<tr>
<td>Interlacing rotation</td>
<td>Float</td>
<td>Interlacing rotation</td>
</tr>
<tr>
<td>Sync wave phase</td>
<td>Float</td>
<td>Sync wave phase</td>
</tr>
<tr>
<td>Sync wave frequency</td>
<td>Float</td>
<td>Sync wave frequency</td>
</tr>
<tr>
<td>Sync wave amplitude</td>
<td>Float</td>
<td>Sync wave amplitude</td>
</tr>
<tr>
<td>Chroma shift</td>
<td>Float</td>
<td>Chromatic shift</td>
</tr>
<tr>
<td>Grain</td>
<td>Float</td>
<td>Amount of image grain</td>
</tr>
<tr>
<td>Color tinting</td>
<td>Vec3</td>
<td>Amount of color tinting</td>
</tr>
<tr>
<td>VisualArtifacts</td>
<td>String</td>
<td>Name of texture used</td>
</tr>
</tbody>
</table>

**ScreenCapture node**

Used to capture a screenshot.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capture</td>
<td>Any</td>
<td>Trigger to capture the screenshot</td>
</tr>
<tr>
<td>FileName</td>
<td>Any</td>
<td>File to write the screenshot capture to</td>
</tr>
<tr>
<td>ImageType</td>
<td>Any</td>
<td>File type to use</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Screenshot capture successful</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Screenshot capture failed</td>
</tr>
</tbody>
</table>
ScreenFader node

Used to perform customizable fade-in and fade-out effects, including the ability to fade from textures. The UseCurColor input uses the previously set color as the fading color if set to True, else it uses the FadeColor value.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FadeGroup</td>
<td>Any</td>
<td>Fade group</td>
</tr>
<tr>
<td>FadeIn</td>
<td>Any</td>
<td>Fade back from the specified color to a normal screen</td>
</tr>
<tr>
<td>FadeOut</td>
<td>Any</td>
<td>Fades the screen to the specified color</td>
</tr>
<tr>
<td>UseCurColor</td>
<td>Boolean</td>
<td>Uses the current color as the source color</td>
</tr>
<tr>
<td>FadeInTime</td>
<td>Float</td>
<td>Duration of fade in</td>
</tr>
<tr>
<td>FadeOutTime</td>
<td>Float</td>
<td>Duration of fade out</td>
</tr>
<tr>
<td>FadeColor</td>
<td>Vec3</td>
<td>Target color to fade to</td>
</tr>
<tr>
<td>TextureName</td>
<td>String</td>
<td>Name of the texture</td>
</tr>
<tr>
<td>UpdateAlways</td>
<td>Boolean</td>
<td>Use to always update the fader</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FadedIn</td>
<td>Any</td>
<td>Triggered when the screen completed faded in</td>
</tr>
<tr>
<td>FadedOut</td>
<td>Any</td>
<td>Triggered when the screen completed faded out</td>
</tr>
<tr>
<td>CurColor</td>
<td>Any</td>
<td>Current faded color</td>
</tr>
</tbody>
</table>

SetShadowMode node

Used to set the shadow mode to Normal or HighQuality mode. Intended to be used for very specific lighting setups and will likely result in self-shadowing artifacts under typical use.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>ShadowMode</td>
<td>Integer</td>
<td>Shadow mode type to use</td>
</tr>
</tbody>
</table>

Input Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to capture input events and configure input settings.

Topics

- ActionFilter node (p. 360)
- ActionHandler node (p. 361)
- ActionListener node (p. 361)
- ActionMapManager node (p. 362)
- Gestures nodes (p. 362)
- MotionSensor nodes (p. 368)
- MouseButtonInfo node (p. 373)
- MouseCoords node (p. 373)
- MouseCursor node (p. 374)
- MouseEntitiesInBox node (p. 374)
- MouseRayCast node (p. 375)
- MouseSetPos node (p. 376)
- Touch:MultiTouchEvent node (p. 376)
- Touch:TouchEvent node (p. 376)
- Touch:MultiTouchCoords node (p. 377)
- Touch:TouchRaycast node (p. 378)
- Touch:VirtualThumbstick node (p. 378)

ActionFilter node

Used to catch key inputs. Should only be used for debugging purposes however.
Lumberyard Legacy Reference
Input Nodes

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Filter</td>
<td>String</td>
<td>Name of the action filter</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Any</td>
<td>Triggers when enabled</td>
</tr>
<tr>
<td>Disabled</td>
<td>Any</td>
<td>Triggers when disabled</td>
</tr>
</tbody>
</table>

ActionHandler node

Used to respond to actions listed in the **Action Map** input.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables listening to the action map</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Enables listening to the action map</td>
</tr>
<tr>
<td>Action Map</td>
<td>String</td>
<td>Name of the action map</td>
</tr>
<tr>
<td>ActionName</td>
<td>String</td>
<td>Name of the action to listen for</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionInvoked</td>
<td>Any</td>
<td>Triggers when the action is invoked</td>
</tr>
<tr>
<td>ActionPressed</td>
<td>Any</td>
<td>Triggers when the action is pressed</td>
</tr>
<tr>
<td>ActionHeld</td>
<td>Any</td>
<td>Triggers when the action is sustained</td>
</tr>
<tr>
<td>ActionReleased</td>
<td>Any</td>
<td>Triggers when the action is released</td>
</tr>
</tbody>
</table>

ActionListener node

Used to listen for action events listed in the **Action Map**.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
<tr>
<td>Action</td>
<td>String</td>
<td>Action to trigger</td>
</tr>
<tr>
<td>Action Map</td>
<td>String</td>
<td>Action map to use</td>
</tr>
<tr>
<td>NonDevMode</td>
<td>Boolean</td>
<td>When set to true, can be used in non dev mode as well</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pressed</td>
<td>String</td>
<td>Triggers when the action is pressed</td>
</tr>
<tr>
<td>Released</td>
<td>String</td>
<td>Triggers when the action is released</td>
</tr>
</tbody>
</table>

**ActionMapManager node**

Used to enable or disable the **Action Map** input.

**Gestures nodes**

This group of nodes is used to handle finger taps, swipes, and other gestures as input.

**Gestures:ClickOrTap node**

Used to recognize one or more mouse clicks or finger taps.
### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
<tr>
<td>PointerIndex</td>
<td>Integer</td>
<td>Pointer (button or finger) index to track</td>
</tr>
<tr>
<td>MinClicksOrTaps</td>
<td>Integer</td>
<td>Minimum number of clicks or taps required for the gesture to be recognized</td>
</tr>
<tr>
<td>MaxSecondsHeld</td>
<td>Float</td>
<td>Maximum number of seconds allowed while held before the gesture stops being recognized</td>
</tr>
<tr>
<td>MaxPixelsMoved</td>
<td>Float</td>
<td>Maximum distance in pixels allowed to move while held before the gesture stops being recognized</td>
</tr>
<tr>
<td>MaxSecondsBetweenClicksOrTaps</td>
<td>Float</td>
<td>Maximum number of seconds allowed between clicks or taps</td>
</tr>
<tr>
<td>MaxPixelsBetweenClicksOrTaps</td>
<td>Float</td>
<td>Maximum distance in pixels allowed between clicks or taps</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recognized</td>
<td>Any</td>
<td>Triggers when a discrete number of clicks or taps is recognized</td>
</tr>
<tr>
<td>StartX</td>
<td>Float</td>
<td>X-axis screen position of the click or tap start</td>
</tr>
<tr>
<td>StartY</td>
<td>Float</td>
<td>Y-axis screen position of the click or tap start</td>
</tr>
<tr>
<td>EndX</td>
<td>Float</td>
<td>X-axis screen position of the click or tap end</td>
</tr>
<tr>
<td>EndY</td>
<td>Float</td>
<td>Y-axis screen position of the click or tap end</td>
</tr>
</tbody>
</table>

### Gestures: Drag node

Used to recognize finger drag gestures.
# Input Nodes

## Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
<tr>
<td>PointerIndex</td>
<td>Integer</td>
<td>Pointer (button or finger) index to track</td>
</tr>
<tr>
<td>MinSecondsHeld</td>
<td>Float</td>
<td>Minimum number of seconds after the initial press before a drag is recognized</td>
</tr>
<tr>
<td>MinPixelsMoved</td>
<td>Float</td>
<td>Minimum distance in pixels before a drag is recognized</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initiated</td>
<td>Any</td>
<td>Activated when a continuous drag gesture is initiated</td>
</tr>
<tr>
<td>Updated</td>
<td>Any</td>
<td>Activated when a continuous drag gesture is updated</td>
</tr>
<tr>
<td>Ended</td>
<td>Any</td>
<td>Activated when a continuous drag gesture has ended</td>
</tr>
<tr>
<td>StartX</td>
<td>Float</td>
<td>X-axis screen position of the drag start</td>
</tr>
<tr>
<td>StartY</td>
<td>Float</td>
<td>X-axis screen position of the drag start</td>
</tr>
<tr>
<td>CurrentX</td>
<td>Float</td>
<td>Current x-axis screen position of the drag</td>
</tr>
<tr>
<td>CurrentY</td>
<td>Float</td>
<td>Current y-axis screen position of the drag</td>
</tr>
<tr>
<td>DeltaX</td>
<td>Float</td>
<td>Number of pixels dragged on the x-axis screen</td>
</tr>
<tr>
<td>DeltaY</td>
<td>Float</td>
<td>Number of pixels dragged on the y-axis screen</td>
</tr>
<tr>
<td>Distance</td>
<td>Float</td>
<td>Number of pixels dragged on screen</td>
</tr>
</tbody>
</table>

---

**Gestures:Hold node**

Used to recognize finger hold gestures.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
<tr>
<td>PointerIndex</td>
<td>Integer</td>
<td>The button or finger index to track</td>
</tr>
<tr>
<td>MinSecondsHeld</td>
<td>Float</td>
<td>Minimum number of seconds before a hold is recognized</td>
</tr>
<tr>
<td>MaxPixelsMoved</td>
<td>Float</td>
<td>Minimum distance in pixels before a hold is recognized</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initiated</td>
<td>Any</td>
<td>Activated when a continuous hold gesture is initiated</td>
</tr>
<tr>
<td>Updated</td>
<td>Any</td>
<td>Activated when a continuous hold gesture is updated</td>
</tr>
<tr>
<td>Ended</td>
<td>Any</td>
<td>Activated when a continuous hold gesture has ended</td>
</tr>
<tr>
<td>StartX</td>
<td>Float</td>
<td>X-axis screen position of the hold start</td>
</tr>
<tr>
<td>StartY</td>
<td>Float</td>
<td>Y-axis screen position of the hold start</td>
</tr>
<tr>
<td>CurrentX</td>
<td>Float</td>
<td>Current x-axis screen position of the hold</td>
</tr>
<tr>
<td>CurrentY</td>
<td>Float</td>
<td>Current y-axis screen position of the hold</td>
</tr>
<tr>
<td>Duration</td>
<td>Float</td>
<td>Duration of the hold in seconds</td>
</tr>
</tbody>
</table>

Gestures:Pinch node

Used to recognize finger pinch (away from or toward) gestures.
## Input Nodes

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
<tr>
<td>MinPixelsMoved</td>
<td>Float</td>
<td>Minimum distance in pixels before a pinch is recognized</td>
</tr>
<tr>
<td>MaxAngleDegrees</td>
<td>Float</td>
<td>Maximum angle in degrees that pinch can deviate before it is recognized</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initiated</td>
<td>Any</td>
<td>Activated when a continuous pinch gesture is initiated</td>
</tr>
<tr>
<td>Updated</td>
<td>Any</td>
<td>Activated when a continuous pinch gesture is updated</td>
</tr>
<tr>
<td>StartMidpointX</td>
<td>Any</td>
<td>Midpoint x-axis position of the pinch</td>
</tr>
<tr>
<td>StartMidpointY</td>
<td>Float</td>
<td>Midpoint y-axis position of the pinch</td>
</tr>
<tr>
<td>StartDistance</td>
<td>Float</td>
<td>Pixel distance between the two touch positions when the pinch is started</td>
</tr>
<tr>
<td>CurrentMidpointX</td>
<td>Float</td>
<td>Current x-axis position of the pinch</td>
</tr>
<tr>
<td>CurrentMidpointY</td>
<td>Float</td>
<td>Current y-axis position of the pinch</td>
</tr>
<tr>
<td>CurrentDistance</td>
<td>Float</td>
<td>Current distance in pixels between the two touch positions</td>
</tr>
<tr>
<td>Ratio</td>
<td>Float</td>
<td>Ratio of the pinch (CurrentDistance/StartDistance)</td>
</tr>
</tbody>
</table>

### Gestures: Rotate node

Used to recognize finger rotation (movement in a circle around each other) gestures.

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
</tbody>
</table>
### Input Nodes

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
<tr>
<td>MaxPixelsMoved</td>
<td>Float</td>
<td>Maximum distance in pixels before a rotation is recognized</td>
</tr>
<tr>
<td>MinAngleDegrees</td>
<td>Float</td>
<td>Minimum angle in degrees before a rotation is recognized</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Port</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>Initiated</td>
<td>Any</td>
<td>Activated when a continuous rotation gesture is initiated</td>
</tr>
<tr>
<td>Updated</td>
<td>Any</td>
<td>Activated when a continuous rotation gesture is updated</td>
</tr>
<tr>
<td>Ended</td>
<td>Any</td>
<td>Activated when a continuous rotation gesture has ended</td>
</tr>
<tr>
<td>StartMidpointX</td>
<td>Float</td>
<td>X-axis screen position where the rotation started</td>
</tr>
<tr>
<td>StartMidpointY</td>
<td>Float</td>
<td>Y-axis screen position where the rotation started</td>
</tr>
<tr>
<td>StartDistance</td>
<td>Float</td>
<td>Pixel distance between the two touch positions when the rotation started</td>
</tr>
<tr>
<td>CurrentMidpointX</td>
<td>Float</td>
<td>Current x-axis screen position of the rotation</td>
</tr>
<tr>
<td>CurrentMidpointY</td>
<td>Float</td>
<td>Current y-axis screen position of the rotation</td>
</tr>
<tr>
<td>CurrentDistance</td>
<td>Float</td>
<td>Current pixel distance between the two touch positions</td>
</tr>
<tr>
<td>RotationDegrees</td>
<td>Float</td>
<td>Current rotation in degrees</td>
</tr>
</tbody>
</table>

### Gestures: Swipe node

Used to recognize finger swipe gestures.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
</tbody>
</table>
Input Nodes

### Port

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
<tr>
<td>PointerIndex</td>
<td>Integer</td>
<td>The button or finger index to track</td>
</tr>
<tr>
<td>MaxSecondsHeld</td>
<td>Float</td>
<td>Maximum number of seconds for a swipe to be recognized</td>
</tr>
<tr>
<td>MinPixelsMoved</td>
<td>Float</td>
<td>Minimum distance in pixels before a swipe is recognized</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recognized</td>
<td>Any</td>
<td>Activated when a continuous swipe gesture is recognized</td>
</tr>
<tr>
<td>StartX</td>
<td>Float</td>
<td>X-axis screen position where the swipe started</td>
</tr>
<tr>
<td>StartY</td>
<td>Float</td>
<td>Y-axis screen position where the swipe started</td>
</tr>
<tr>
<td>EndX</td>
<td>Float</td>
<td>X-axis screen position where the swipe ended</td>
</tr>
<tr>
<td>EndY</td>
<td>Float</td>
<td>Y-axis screen position where the swipe ended</td>
</tr>
<tr>
<td>DeltaX</td>
<td>Float</td>
<td>X-axis pixels swiped</td>
</tr>
<tr>
<td>DeltaY</td>
<td>Float</td>
<td>Y-axis pixels swiped</td>
</tr>
<tr>
<td>DirectionX</td>
<td>Float</td>
<td>X-axis direction of the swipe</td>
</tr>
<tr>
<td>DirectionY</td>
<td>Float</td>
<td>Y-axis direction of the swipe</td>
</tr>
<tr>
<td>Distance</td>
<td>Float</td>
<td>Distance of the swipe in pixels</td>
</tr>
<tr>
<td>Duration</td>
<td>Float</td>
<td>Duration of the swipe in seconds</td>
</tr>
<tr>
<td>Velocity</td>
<td>Float</td>
<td>Velocity of the swipe in pixels per second</td>
</tr>
</tbody>
</table>

### MotionSensor nodes

This group of nodes are used with a motion sensor or accelerometer input.

**MotionSensor:AccelerationGravity node**

Used to output gravity-generated acceleration.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SensorData</td>
<td>Vec3</td>
<td>Outputs raw gravity acceleration in g-forces</td>
</tr>
<tr>
<td>IsSensorDataAvailable</td>
<td>Boolean</td>
<td>Outputs true or false when the node is activated</td>
</tr>
</tbody>
</table>

**MotionSensor:AccelerationRaw node**

Used to output raw acceleration.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SensorData</td>
<td>Vec3</td>
<td>Outputs raw sensor acceleration in g-forces</td>
</tr>
<tr>
<td>IsSensorDataAvailable</td>
<td>Boolean</td>
<td>Outputs true or false when the node is activated</td>
</tr>
</tbody>
</table>

**MotionSensor:AccelerationUser node**

Used to output user-generated acceleration.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SensorData</td>
<td>Vec3</td>
<td>Outputs user-generated acceleration in g-forces</td>
</tr>
<tr>
<td>IsSensorDataAvailable</td>
<td>Boolean</td>
<td>Outputs true or false when the node is activated</td>
</tr>
</tbody>
</table>
MotionSensor:MagneticFieldRaw node

Used to output raw magnetic field data as measured by a magnetometer. Includes device bias.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SensorData</td>
<td>Vec3</td>
<td>Outputs raw magnetic field in microteslas</td>
</tr>
<tr>
<td>IsSensorDataAvailable</td>
<td>Boolean</td>
<td>Outputs true or false when the node is activated</td>
</tr>
</tbody>
</table>

MotionSensor:MagneticFieldUnbiased node

Used to output magnetic field data as measured by a magnetometer. Processed to remove device bias.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SensorData</td>
<td>Vec3</td>
<td>Outputs unbiased magnetic field data in microteslas</td>
</tr>
<tr>
<td>IsSensorDataAvailable</td>
<td>Boolean</td>
<td>Outputs true or false when the node is activated</td>
</tr>
</tbody>
</table>

MotionSensor:MagneticNorth node

Used to output a vector pointing to magnetic north.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SensorData</td>
<td>Vec3</td>
<td>Outputs a vector pointing to magnetic north</td>
</tr>
<tr>
<td>IsSensorDataAvailable</td>
<td>Boolean</td>
<td>Outputs true or false when the node is activated</td>
</tr>
</tbody>
</table>

MotionSensor:Orientation node

Used to measure the orientation or attitude of the device from an arbitrary but constant frame of reference.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
</tbody>
</table>

MotionSensor:OrientationDelta node

Used to measure the change in orientation or attitude of the device since the last measurement.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SensorData</td>
<td>Vec3</td>
<td>Outputs an orientation or attitude angle in degrees</td>
</tr>
<tr>
<td>IsSensorDataAvailable</td>
<td>Boolean</td>
<td>Outputs true or false when the node is activated</td>
</tr>
</tbody>
</table>
# Input Nodes

## Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SensorData</td>
<td>Vec3</td>
<td>Outputs an orientation or attitude angle in degrees</td>
</tr>
<tr>
<td>IsSensorDataAvailable</td>
<td>Boolean</td>
<td>Outputs true or false when the node is activated</td>
</tr>
</tbody>
</table>

## MotionSensor:RotationRateRaw node

Used to output the raw rotation rate as measured by the gyroscope.

## Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SensorData</td>
<td>Vec3</td>
<td>Outputs the raw gyroscope rotation rate in degrees per second</td>
</tr>
<tr>
<td>IsSensorDataAvailable</td>
<td>Boolean</td>
<td>Outputs true or false when the node is activated</td>
</tr>
</tbody>
</table>

## MotionSensor:RotationRateUnbiased node

Used to output the rotation rate as measured by the gyroscope and processed to remove device bias.

## Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SensorData</td>
<td>Vec3</td>
<td>Outputs an unbiased rotation rate in degrees per second</td>
</tr>
<tr>
<td>Port</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IsSensorDataAvailable</td>
<td>Boolean</td>
<td>Outputs true or false when the node is activated</td>
</tr>
</tbody>
</table>

**MouseButtonInfo node**

Used to output mouse button state information.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
<tr>
<td>MouseButton</td>
<td>Boolean</td>
<td>Mouse button state information</td>
</tr>
<tr>
<td>MouseWheel</td>
<td>Boolean</td>
<td>Mouse wheel state information</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MousePressed</td>
<td>Integer</td>
<td>Outputs the mouse button that was pressed</td>
</tr>
<tr>
<td>MouseReleased</td>
<td>Integer</td>
<td>Outputs the mouse button that was released</td>
</tr>
<tr>
<td>MouseWheel</td>
<td>Float</td>
<td>Outputs a positive value when the mouse wheel is moved up and a negative value when moved down</td>
</tr>
</tbody>
</table>

**MouseCoords node**

Used to output mouse coordinates.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
<tr>
<td>World</td>
<td>Boolean</td>
<td>World coordinates used</td>
</tr>
<tr>
<td>Screen</td>
<td>Boolean</td>
<td>Screen coordinates of the mouse cursor</td>
</tr>
</tbody>
</table>
### Input Nodes

#### Port Type Description

**Delta** Boolean Shows the number of screen pixels the mouse cursor has moved

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>World</td>
<td>Vec3</td>
<td>World coordinates of the mouse cursor</td>
</tr>
<tr>
<td>ScreenX</td>
<td>Integer</td>
<td>X-axis coordinate of mouse cursor</td>
</tr>
<tr>
<td>ScreenY</td>
<td>Integer</td>
<td>Y-axis coordinate of mouse cursor</td>
</tr>
<tr>
<td>DeltaScreenX</td>
<td>Integer</td>
<td>X-axis delta coordinate of mouse cursor</td>
</tr>
<tr>
<td>DeltaScreenY</td>
<td>Integer</td>
<td>Y-axis delta coordinate of mouse cursor</td>
</tr>
</tbody>
</table>

#### MouseCursor node

Used to show or hide the mouse cursor.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show</td>
<td>Any</td>
<td>Shows the mouse cursor</td>
</tr>
<tr>
<td>Hide</td>
<td>Any</td>
<td>Hides the mouse cursor</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when the action is complete</td>
</tr>
</tbody>
</table>

#### MouseEntitiesInBox node

Used to show or hide the mouse coordinates.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get</td>
<td>Any</td>
<td>Get the mouse cursor</td>
</tr>
</tbody>
</table>
### Input Nodes

#### Port

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContainerId</td>
<td>Integer</td>
<td>ID of the container that stores the entities</td>
</tr>
<tr>
<td>ScreenX</td>
<td>Integer</td>
<td>X-axis screen position of the mouse cursor</td>
</tr>
<tr>
<td>ScreenY</td>
<td>Integer</td>
<td>Y-axis screen position of the mouse cursor</td>
</tr>
<tr>
<td>ScreenX2</td>
<td>Integer</td>
<td>X-axis screen position 2 of the mouse cursor</td>
</tr>
<tr>
<td>ScreenY2</td>
<td>Integer</td>
<td>Y-axis screen position 2 of the mouse cursor</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when completed</td>
</tr>
</tbody>
</table>

### MouseRayCast node

Used to output the mouse raycast information.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
<tr>
<td>All</td>
<td>Integer</td>
<td>Raycast filter type</td>
</tr>
<tr>
<td>EntitiesToIgnore</td>
<td>Integer</td>
<td>Entities to ignore during raycast</td>
</tr>
<tr>
<td>IgnoreBackFaces</td>
<td>Boolean</td>
<td>Ignore backfaces of geometry during raycast</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HitPos</td>
<td>Vec3</td>
<td>Coordinates of the first position that was hit with the raycast</td>
</tr>
<tr>
<td>HitNormal</td>
<td>Vec3</td>
<td>Normal of the first position that was hit with the raycast</td>
</tr>
<tr>
<td>EntityId</td>
<td>Any</td>
<td>ID of the entity that was hit by the raycast</td>
</tr>
<tr>
<td>NoHit</td>
<td>Any</td>
<td>Activated each frame when enabled and no item was hit by the raycast</td>
</tr>
</tbody>
</table>
MouseSetPos node

Used to position the mouse at the specified location when activated.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Coords</td>
<td>Vec3</td>
<td>Coordinates to set the mouse at</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Any</td>
<td>Triggers when the new mouse position is set</td>
</tr>
</tbody>
</table>

Touch:MultiTouchEvent node

Used to output finger touch location.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TouchDown</td>
<td>Integer</td>
<td>Finger (touch) ID that was pressed</td>
</tr>
<tr>
<td>TouchUp</td>
<td>Integer</td>
<td>Finger (touch) ID that was released</td>
</tr>
</tbody>
</table>

Touch:TouchEvent node

Used to output finger touch location.
## Input Nodes

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
<tr>
<td>TouchId</td>
<td>Integer</td>
<td>Touch (finger) ID for which events will be sent from</td>
</tr>
<tr>
<td>ScreenCoords</td>
<td>Boolean</td>
<td>Output screen coordinates</td>
</tr>
<tr>
<td>WorldCoords</td>
<td>Boolean</td>
<td>Output world coordinates</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TouchDown</td>
<td>Any</td>
<td>Finger (touch) ID that was pressed</td>
</tr>
<tr>
<td>TouchUp</td>
<td>Any</td>
<td>Finger (touch) ID that was released</td>
</tr>
<tr>
<td>ScreenCoordX</td>
<td>Integer</td>
<td>Screen x-axis coordinate of the touch</td>
</tr>
<tr>
<td>ScreenCoordY</td>
<td>Integer</td>
<td>Screen y-axis coordinate of the touch</td>
</tr>
<tr>
<td>WorldCoords</td>
<td>Vec3</td>
<td>Touch position in world coordinates</td>
</tr>
</tbody>
</table>

### Touch:MultiTouchCoords node

Used to output the finger touch location from the specified ID.

<table>
<thead>
<tr>
<th>Inputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
<tr>
<td>TouchId</td>
<td>Integer</td>
<td>Finger (touch) ID for which the coordinates will obtained</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ScreenCoordX</td>
<td>Integer</td>
<td>X-axis location of the finger touch</td>
</tr>
<tr>
<td>ScreenCoordY</td>
<td>Integer</td>
<td>Y-axis location of the finger touch</td>
</tr>
</tbody>
</table>
**Touch:TouchRaycast node**

Used to generate a raycast for each finger frame ID.

![Input:Touch:TouchRaycast node](image)

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
<tr>
<td>All</td>
<td>Integer</td>
<td>Raycast filter type</td>
</tr>
<tr>
<td>EntitiesToIgnore</td>
<td>Integer</td>
<td>Entities to ignore during raycast</td>
</tr>
<tr>
<td>IgnoreBackFaces</td>
<td>Boolean</td>
<td>Ignore backfaces of geometry during raycast</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HitPos</td>
<td>Vec3</td>
<td>Coordinates of the first position that was hit with the raycast</td>
</tr>
<tr>
<td>HitNormal</td>
<td>Vec3</td>
<td>Normal of the first position that was hit with the raycast</td>
</tr>
<tr>
<td>EntityId</td>
<td>Any</td>
<td>ID of the entity that was hit by the raycast</td>
</tr>
<tr>
<td>NoHit</td>
<td>Any</td>
<td>Activated each frame when enabled and no item was hit by the raycast</td>
</tr>
</tbody>
</table>

**Touch:VirtualThumbstick node**

Used to implement a virtual thumbstick.

![Input:Touch:VirtualThumbstick node](image)

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the node</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the node</td>
</tr>
<tr>
<td>Radius</td>
<td>Float</td>
<td>Radius of thumbstick pad as a percentage of screen width</td>
</tr>
</tbody>
</table>
### Interpolate Nodes

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use these flow graph nodes to configure interpolate-related settings.

#### Topics
- Color node (p. 379)
- Float node (p. 380)
- Int node (p. 381)
- SmoothAngleVec3 (p. 381)
- SmoothColor node (p. 382)
- SmoothFloat node (p. 382)
- SmoothInt node (p. 383)
- SmoothVec3 node (p. 383)
- Vec3 node (p. 384)

### Color node

Used to linearly calculate from an initial color value to an end color value within a given time frame.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Starts interpolation</td>
</tr>
<tr>
<td>Stop</td>
<td>Any</td>
<td>Stops interpolation</td>
</tr>
<tr>
<td>StartValue</td>
<td>Vec3</td>
<td>Starting value for color</td>
</tr>
<tr>
<td>EndValue</td>
<td>Vec3</td>
<td>Ending value for color</td>
</tr>
<tr>
<td>Time</td>
<td>Float</td>
<td>Interpolation duration in seconds</td>
</tr>
<tr>
<td>UpdateFrequency</td>
<td>Float</td>
<td>Interpolation update frequency in seconds. 0 = every frame</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Vec3</td>
<td>Current value</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when finished</td>
</tr>
</tbody>
</table>

**Float node**

Used to linearly calculate from an initial floating point value to an end floating point value within a given time frame.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Starts interpolation</td>
</tr>
<tr>
<td>Stop</td>
<td>Any</td>
<td>Stops interpolation</td>
</tr>
<tr>
<td>StartValue</td>
<td>Float</td>
<td>Starting value for floating point</td>
</tr>
<tr>
<td>EndValue</td>
<td>Float</td>
<td>Ending value for floating point</td>
</tr>
<tr>
<td>Time</td>
<td>Float</td>
<td>Interpolation duration in seconds</td>
</tr>
<tr>
<td>UpdateFrequency</td>
<td>Float</td>
<td>Interpolation update frequency in seconds. 0 = every frame</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Float</td>
<td>Current value</td>
</tr>
</tbody>
</table>
**Int node**

Used to linearly calculate from an initial integer value to an end integer value within a given time frame.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Starts interpolation</td>
</tr>
<tr>
<td>Stop</td>
<td>Any</td>
<td>Stops interpolation</td>
</tr>
<tr>
<td>StartValue</td>
<td>Integer</td>
<td>Starting value for integer</td>
</tr>
<tr>
<td>EndValue</td>
<td>Integer</td>
<td>Ending value for integer</td>
</tr>
<tr>
<td>Time</td>
<td>Float</td>
<td>Interpolation duration in seconds</td>
</tr>
<tr>
<td>UpdateFrequency</td>
<td>Float</td>
<td>Interpolation update frequency in seconds. 0 = every frame</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Integer</td>
<td>Current value</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when finished</td>
</tr>
</tbody>
</table>

**SmoothAngleVec3**

Used to non-linearly (damped spring system) calculate from an initial vector angle to an end vector angle within a given time frame. Calculation will slow down as it reaches the end value.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>InitialValue</td>
<td>Vec3</td>
<td>Initial interpolation value for vector angle</td>
</tr>
<tr>
<td>TargetValue</td>
<td>Vec3</td>
<td>Target interpolation value for vector angle</td>
</tr>
</tbody>
</table>
### SmoothColor node

Used to non-linearly (damped spring system) calculate from an initial color value to an end color value within a given time frame. Calculation will slow down as it reaches the end value.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>InitialValue</td>
<td>Vec3</td>
<td>Initial interpolation value for color</td>
</tr>
<tr>
<td>TargetValue</td>
<td>Vec3</td>
<td>Target interpolation value for color</td>
</tr>
<tr>
<td>Time</td>
<td>Float</td>
<td>Interpolation duration in seconds</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Vec3</td>
<td>Current value</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when finished</td>
</tr>
</tbody>
</table>

### SmoothFloat node

Used to non-linearly (damped spring system) calculate from an initial floating point value to an end floating point value within a given time frame. Calculation will slow down as it reaches the end value.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Vec3</td>
<td>Current value</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when finished</td>
</tr>
</tbody>
</table>
### Interpolate Nodes

#### InitialValue

**Port**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitialValue</td>
<td>Float</td>
<td>Initial interpolation value for floating point</td>
</tr>
<tr>
<td>TargetValue</td>
<td>Float</td>
<td>Target interpolation value for floating point</td>
</tr>
<tr>
<td>Time</td>
<td>Float</td>
<td>Interpolation duration in seconds</td>
</tr>
</tbody>
</table>

#### Value

**Port**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Float</td>
<td>Current value</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when finished</td>
</tr>
</tbody>
</table>

#### SmoothInt node

Used to non-linearly (damped spring system) calculate from an initial integer value to an end integer value within a given time frame. Calculation will slow down as it reaches the end value.

#### Inputs

**Port**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>InitialValue</td>
<td>Integer</td>
<td>Initial interpolation value for integer</td>
</tr>
<tr>
<td>TargetValue</td>
<td>Integer</td>
<td>Target interpolation value for integer</td>
</tr>
<tr>
<td>Time</td>
<td>Float</td>
<td>Interpolation duration in seconds</td>
</tr>
</tbody>
</table>

#### Outputs

**Port**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Integer</td>
<td>Current value</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when finished</td>
</tr>
</tbody>
</table>

#### SmoothVec3 node

Used to non-linearly (damped spring system) calculate from an initial Vec3 value to an end Vec3 value within a given time frame. Calculation will slow down as it reaches the end value.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>InitialValue</td>
<td>Vec3</td>
<td>Initial interpolation value for Vec3</td>
</tr>
<tr>
<td>TargetValue</td>
<td>Vec3</td>
<td>Target interpolation value for Vec3</td>
</tr>
<tr>
<td>Time</td>
<td>Float</td>
<td>Interpolation duration in seconds</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Vec3</td>
<td>Current value</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when finished</td>
</tr>
</tbody>
</table>

Vec3 node

Used to linearly calculate from an initial Vec3 value to an end Vec3 value within a given time frame.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Starts interpolation</td>
</tr>
<tr>
<td>Stop</td>
<td>Any</td>
<td>Stops interpolation</td>
</tr>
<tr>
<td>StartValue</td>
<td>Vec3</td>
<td>Starting value for Vec3</td>
</tr>
<tr>
<td>EndValue</td>
<td>Vec3</td>
<td>Ending value for Vec3</td>
</tr>
<tr>
<td>Time</td>
<td>Float</td>
<td>Interpolation duration in seconds</td>
</tr>
<tr>
<td>UpdateFrequency</td>
<td>Float</td>
<td>Interpolation update frequency in seconds. 0 = every frame</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Vec3</td>
<td>Current value</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when finished</td>
</tr>
</tbody>
</table>
Intersection Tests Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to configure intersection tests.

Topics
- BoundingBoxVsBoundingBox node (p. 385)
- BoundingBoxVsSphere node (p. 385)

BoundingBoxVsBoundingBox node

Used to test two bounding boxes to see if they intersect.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port</td>
<td>Type</td>
</tr>
<tr>
<td>Activate</td>
<td>Any</td>
</tr>
<tr>
<td>Min1</td>
<td>Vec3</td>
</tr>
<tr>
<td>Max1</td>
<td>Vec3</td>
</tr>
<tr>
<td>Min2</td>
<td>Vec3</td>
</tr>
<tr>
<td>Max2</td>
<td>Vec3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port</td>
<td>Type</td>
</tr>
<tr>
<td>Result</td>
<td>Boolean</td>
</tr>
<tr>
<td>True</td>
<td>Any</td>
</tr>
<tr>
<td>False</td>
<td>Any</td>
</tr>
</tbody>
</table>

BoundingBoxVsSphere node

Used to test a bounding box and a sphere to see if they intersect.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>BoundsMin</td>
<td>Vec3</td>
<td>Minimum point of the bounding box</td>
</tr>
<tr>
<td>BoundsMax</td>
<td>Vec3</td>
<td>Maximum point of the bounding box</td>
</tr>
<tr>
<td>SphereCenter</td>
<td>Vec3</td>
<td>Center of the sphere</td>
</tr>
<tr>
<td>SphereRadius</td>
<td>Float</td>
<td>Radius of the sphere</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Boolean</td>
<td>Outputs true if an intersection occurred</td>
</tr>
<tr>
<td>True</td>
<td>Any</td>
<td>Triggers if an intersection occurred</td>
</tr>
<tr>
<td>False</td>
<td>Any</td>
<td>Triggers if an intersection did not occur</td>
</tr>
</tbody>
</table>

**Iterator Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about [Script Canvas](#), Lumberyard’s new visual scripting environment.

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 about the latest features, see the [Amazon Lumberyard User Guide](#).

You can use the following flow graph nodes to configure iterator-related settings.

**Topics**

- GetEntities node (p. 386)
- GetEntitiesInArea node (p. 387)
- GetEntitiesInBox node (p. 388)
- GetEntitiesInSphere node (p. 389)

**GetEntities node**

Used to find and return all entities in the world.
## Lumberyard Legacy Reference
### Iterator Nodes

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>Next</td>
<td>Any</td>
<td>Gets the next entity found</td>
</tr>
<tr>
<td>Limit</td>
<td>Integer</td>
<td>Limits how many entities are returned</td>
</tr>
<tr>
<td>Immediate</td>
<td>Boolean</td>
<td>Iterates immediately through the results</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
<td>Type of entity to iterate</td>
</tr>
<tr>
<td>ArchetypeFilter</td>
<td>Any</td>
<td>Returns archetype entities</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutEntityId</td>
<td>Any</td>
<td>Outputs the entity and entity ID</td>
</tr>
<tr>
<td>Count</td>
<td>Integer</td>
<td>Outputs the current of entities</td>
</tr>
<tr>
<td>Done</td>
<td>Integer</td>
<td>Triggered when all entities have been found, with the total count returned</td>
</tr>
</tbody>
</table>

### GetEntitiesInArea node

Used to find and return all entities within the specified area shape.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>Next</td>
<td>Any</td>
<td>Gets the next entity found</td>
</tr>
<tr>
<td>Limit</td>
<td>Integer</td>
<td>Limits how many entities are returned</td>
</tr>
<tr>
<td>Immediate</td>
<td>Boolean</td>
<td>Iterates immediately through the results</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
<td>Type of entity to iterate</td>
</tr>
<tr>
<td>ArchetypeFilter</td>
<td>String</td>
<td>Returns archetype entities</td>
</tr>
</tbody>
</table>
**Iterator Nodes**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area</td>
<td>String</td>
<td>Name of area shape to test against</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutEntityId</td>
<td>Any</td>
<td>Outputs the entity and entity ID</td>
</tr>
<tr>
<td>Count</td>
<td>Integer</td>
<td>Outputs the current of entities</td>
</tr>
<tr>
<td>Done</td>
<td>Integer</td>
<td>Triggered when all entities have been found, with the total count returned</td>
</tr>
</tbody>
</table>

**GetEntitiesInBox node**

Used to find and return all entities within the defined AABB box.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>Next</td>
<td>Any</td>
<td>Gets the next entity found</td>
</tr>
<tr>
<td>Limit</td>
<td>Integer</td>
<td>Limits how many entities are returned</td>
</tr>
<tr>
<td>Immediate</td>
<td>Any</td>
<td>Iterates immediately through the results</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
<td>Type of entity to iterate</td>
</tr>
<tr>
<td>ArchetypeFilter</td>
<td>String</td>
<td>Returns archetype entities</td>
</tr>
<tr>
<td>Min</td>
<td>Vec3</td>
<td>Minimum vector extents of the AABB bounding box to check for entities</td>
</tr>
<tr>
<td>Max</td>
<td>Vec3</td>
<td>Maximum vector extents of the AABB bounding box to check for entities</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutEntityId</td>
<td>Any</td>
<td>Outputs the entity and entity ID</td>
</tr>
<tr>
<td>Count</td>
<td>Integer</td>
<td>Outputs the current of entities</td>
</tr>
</tbody>
</table>
### Port | Type | Description
--- | --- | ---
Done | Integer | Triggered when all entities have been found, with the total count returned

### GetEntitiesInSphere node

Used to find and return all entities within the defined sphere volume.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>Next</td>
<td>Any</td>
<td>Gets the next entity found</td>
</tr>
<tr>
<td>Limit</td>
<td>Integer</td>
<td>Limits how many entities are returned</td>
</tr>
<tr>
<td>Immediate</td>
<td>Boolean</td>
<td>Iterates immediately through the results</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
<td>Type of entity to iterate</td>
</tr>
<tr>
<td>ArchetypeFilter</td>
<td>String</td>
<td>Returns archetype entities</td>
</tr>
<tr>
<td>Center</td>
<td>Vec3</td>
<td>Center of the sphere</td>
</tr>
<tr>
<td>Radius</td>
<td>Float</td>
<td>Distance from the center of the sphere to check for entities</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutEntityId</td>
<td>Any</td>
<td>Outputs the entity and entity ID</td>
</tr>
<tr>
<td>Count</td>
<td>Integer</td>
<td>Outputs the current of entities</td>
</tr>
<tr>
<td>Done</td>
<td>Integer</td>
<td>Triggered when all entities have been found, with the total count returned</td>
</tr>
</tbody>
</table>

### JSON Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about [Script Canvas](https://docs.lumberyardgames.com/Guides/script-canvas-guide), Lumberyard's new visual scripting environment.
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 about the latest features, see the Amazon Lumberyard User Guide.

You can use these flow graph nodes to configure JSON settings.

**Topics**
- GetJsonProperty node (p. 390)
- IsValueInJsonArray node (p. 390)
- IterateJsonArrayProperty node (p. 391)
- SetJsonProperty node (p. 391)

**GetJsonProperty node**

Used to get the JSON attribute value.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>JSON</td>
<td>String</td>
<td>The JSON code to parse</td>
</tr>
<tr>
<td>Attribute</td>
<td>String</td>
<td>The attribute to get the value of</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>String</td>
<td>Triggers if the JSON could not be parsed or the attribute could not be found</td>
</tr>
<tr>
<td>OutValue</td>
<td>String</td>
<td>Outputs the attribute value</td>
</tr>
</tbody>
</table>

**IsValueInJsonArray node**

Used to look through a JSON array for the specified value.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>JsonArray</td>
<td>String</td>
<td>The JSON array to search on</td>
</tr>
<tr>
<td>Port</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Value</td>
<td>String</td>
<td>The JSON value to search for</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Integer</td>
<td>Outputs the number of occurrences found</td>
</tr>
<tr>
<td>True</td>
<td>Boolean</td>
<td>Triggers if the value was found</td>
</tr>
<tr>
<td>False</td>
<td>Boolean</td>
<td>Triggers if the value was not found</td>
</tr>
</tbody>
</table>

**IterateJsonArrayProperty node**

Used to iterate through a JSON array, returning one element at a time.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Begin</td>
<td>Any</td>
<td>Starts iterating over the supplied JSON array</td>
</tr>
<tr>
<td>Continue</td>
<td>Any</td>
<td>Continues iterating over the supplied JSON array</td>
</tr>
<tr>
<td>JsonArray</td>
<td>String</td>
<td>The JSON array to iterated over</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Float</td>
<td>Value of the current array element</td>
</tr>
<tr>
<td>Index</td>
<td>Integer</td>
<td>Index of the current array element</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when there are no more elements in the array</td>
</tr>
<tr>
<td>IsEmpty</td>
<td>Boolean</td>
<td>Triggers if the array is empty</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Triggers if an error occurs</td>
</tr>
</tbody>
</table>

**SetJsonProperty node**

Used to set a property on a JSON object.
Kinect Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard’s new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use these flow graph nodes to configure Kinect settings.

Topics
- Alignment node (p. 392)
- Skeleton node (p. 393)

Alignment node

Used to get the default Kinect skeleton joint lengths when a new closest tracked skeleton is detected.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables the skeleton alignment watcher</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables the skeleton alignment watcher</td>
</tr>
<tr>
<td>ForceAlign</td>
<td>Any</td>
<td>Forces the beginning of the new skeleton alignment phase</td>
</tr>
</tbody>
</table>

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>JsonObject</td>
<td>String</td>
<td>The JSON object to set the property on</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>Name of the JSON property</td>
</tr>
<tr>
<td>Value</td>
<td>String</td>
<td>Value of the JSON property</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>String</td>
<td>Outputs the JSON object</td>
</tr>
</tbody>
</table>
### Logic Nodes

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlignTime</td>
<td>Float</td>
<td>The time spent each time the skeleton alignment is started</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Started</td>
<td>Boolean</td>
<td>Triggers when a new skeleton alignment has started</td>
</tr>
<tr>
<td>Completed</td>
<td>Boolean</td>
<td>Triggers when a skeleton alignment completes</td>
</tr>
</tbody>
</table>

#### Skeleton node

Used to get the status of the joints for a Kinect skeleton.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Auto</td>
<td>Boolean</td>
<td>Forces an auto update</td>
</tr>
<tr>
<td>Freq</td>
<td>Float</td>
<td>Auto update frequency. Use 0 to update every frame.</td>
</tr>
<tr>
<td>Joint</td>
<td>Integer</td>
<td>The skeleton joint</td>
</tr>
<tr>
<td>RefJoint</td>
<td>Integer</td>
<td>The skeleton reference joint</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>Vec3</td>
<td>Outputs the skeleton vector position</td>
</tr>
<tr>
<td>X</td>
<td>Float</td>
<td>Outputs the skeleton x-axis position</td>
</tr>
<tr>
<td>Y</td>
<td>Float</td>
<td>Outputs the skeleton y-axis position</td>
</tr>
<tr>
<td>Z</td>
<td>Float</td>
<td>Outputs the skeleton z-axis position</td>
</tr>
</tbody>
</table>

**Logic Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about [Script Canvas](https://rumble.com/), Lumberyard’s new visual scripting environment.
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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to define logic operations.

**Topics**
- AND node (p. 394)
- All node (p. 395)
- Any node (p. 395)
- Blocker node (p. 396)
- CountBlocker node (p. 396)
- DeMultiplexer node (p. 397)
- Gate node (p. 397)
- IfCondition node (p. 398)
- Indexer node (p. 398)
- Multiplexer node (p. 399)
- NOT node (p. 399)
- OR node (p. 400)
- OnChange node (p. 400)
- Once node (p. 400)
- OnceNoSerialize node (p. 401)
- RandomSelect node (p. 401)
- RandomTrigger node (p. 402)
- SelectCondition node (p. 403)
- Sequencer node (p. 403)
- XOR node (p. 404)

**AND node**

Used to perform a logical AND operation on the input ports. Output is true if both inputs are true.

![AND node diagram]

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>A</td>
<td>Boolean</td>
<td>First input</td>
</tr>
<tr>
<td>B</td>
<td>Boolean</td>
<td>Second input</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Boolean</td>
<td>Output value</td>
</tr>
<tr>
<td>OnTrue</td>
<td>Boolean</td>
<td>Triggers if Out is true</td>
</tr>
<tr>
<td>OnFalse</td>
<td>Boolean</td>
<td>Triggers if Out is false</td>
</tr>
</tbody>
</table>

All node

Used to trigger the output when all connected inputs are triggered.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In0 - In7</td>
<td>Any</td>
<td>Input values</td>
</tr>
<tr>
<td>Reset</td>
<td>Any</td>
<td>Resets the input values to 0</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Any</td>
<td>Triggered when all inputs are triggered</td>
</tr>
</tbody>
</table>

Any node

Used to trigger the output when any of the connected inputs are triggered.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In0 - In9</td>
<td>Any</td>
<td>Input values</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Any</td>
<td>Triggered when any inputs are triggered</td>
</tr>
</tbody>
</table>

**Blocker node**

Used to block or pass signals depending on the state of the Block input.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block</td>
<td>Boolean</td>
<td>If true, blocks In signal</td>
</tr>
<tr>
<td>In</td>
<td>Any</td>
<td>Input signal</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Any</td>
<td>If Block is false, outputs In signal. If Block is true, In signal is blocked.</td>
</tr>
</tbody>
</table>

**CountBlocker node**

Used to output a signal a number of times as defined by the Limit input.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Any</td>
<td>Input value</td>
</tr>
<tr>
<td>Reset</td>
<td>Any</td>
<td>Resets In to 0</td>
</tr>
<tr>
<td>Limit</td>
<td>Integer</td>
<td>Number of times In is sent to Out</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Any</td>
<td>Passes In a limited number of times as defined by Limit</td>
</tr>
</tbody>
</table>
DeMultiplexer node

Used to send the In input to the selected Out output, based on the value of the Mode input:

- **Always**: Both the In and Index inputs activate the output.
- **IndexOnly**: Only the Index input activates the output.
- **InputOnly**: Only the In port activates the output.

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>Integer</td>
<td>Determines which output receives the input (In)</td>
</tr>
<tr>
<td>Mode</td>
<td>Integer</td>
<td>Determines when the outputs are activated</td>
</tr>
<tr>
<td>In</td>
<td>Any</td>
<td>Input value</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out0 - Out7</td>
<td>Any</td>
<td>Outputs that can be triggered</td>
</tr>
</tbody>
</table>

Gate node

Used to block or pass a signal depending on the state of the Closed input.

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Any</td>
<td>Input value</td>
</tr>
<tr>
<td>Closed</td>
<td>Boolean</td>
<td>If true, blocks the input from passing to the output</td>
</tr>
<tr>
<td>Open</td>
<td>Any</td>
<td>Sets Closed to false</td>
</tr>
<tr>
<td>Close</td>
<td>Any</td>
<td>Sets Closed to true</td>
</tr>
</tbody>
</table>
## Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Any</td>
<td>Output value</td>
</tr>
<tr>
<td>OutClosed</td>
<td>Any</td>
<td>Output if Closed is true</td>
</tr>
</tbody>
</table>

### IfCondition node

Used to output signals based on whether the Condition input is enabled.

![Image of IfCondition node]

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>Condition</td>
<td>Boolean</td>
<td>Condition value</td>
</tr>
<tr>
<td>In</td>
<td>Any</td>
<td>Input value</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnFalse</td>
<td>Any</td>
<td>Triggers if Condition is false</td>
</tr>
<tr>
<td>OnTrue</td>
<td>Any</td>
<td>Triggers if Condition is true</td>
</tr>
</tbody>
</table>

### Indexer node

Used to return the index of an active input. Does not account for multiple activations on different inputs.

![Image of Indexer node]

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In0 - In7</td>
<td>Any</td>
<td>Input values</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutIndex</td>
<td>Integer</td>
<td>Outputs the index (number) of the active input</td>
</tr>
</tbody>
</table>
Multiplexer node

Used to select an input and send it to the output, based on the value of the Mode input:

- **Always**: Both the In and Index inputs activate the output.
- **IndexOnly**: Only the Index input activates the output.
- **InputOnly**: Only the In port activates the output.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>Integer</td>
<td>Determines which input is passed to the output</td>
</tr>
<tr>
<td>Mode</td>
<td>Integer</td>
<td>Determines which inputs activate the output</td>
</tr>
<tr>
<td>In0 - In7</td>
<td>Any</td>
<td>Input values</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Any</td>
<td>Output value</td>
</tr>
</tbody>
</table>

NOT node

Used to perform a logical NOT operation on the input ports. If the input is true, the output will be false and vice versa.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Boolean</td>
<td>Input value</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Boolean</td>
<td>If the input is true, the output will be false and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>vice versa</td>
</tr>
</tbody>
</table>
**OR node**

Used to perform a logical OR operation on the input ports. The output is true if either of the two inputs is true.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>A</td>
<td>Boolean</td>
<td>First input</td>
</tr>
<tr>
<td>B</td>
<td>Boolean</td>
<td>Second input</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Boolean</td>
<td>Output is true if either of the two inputs is true</td>
</tr>
<tr>
<td>OnTrue</td>
<td>Boolean</td>
<td>Triggers if Out is true</td>
</tr>
<tr>
<td>OnFalse</td>
<td>Boolean</td>
<td>Triggers if Out is false</td>
</tr>
</tbody>
</table>

**OnChange node**

Used to send the input value to the output when it is different from the previous value.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Boolean</td>
<td>Input value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Boolean</td>
<td>Receives the input value when the input has changed from it's previous value</td>
</tr>
</tbody>
</table>

**Once node**

Used to pass the activated input to the output only once.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In0 - In7</td>
<td>Any</td>
<td>Input values</td>
</tr>
<tr>
<td>Reset</td>
<td>Any</td>
<td>Resets the inputs and allows new activation to occur</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Any</td>
<td>Receives the active input only once</td>
</tr>
</tbody>
</table>

OnceNoSerialize node

Use to pass the activated input value to the output only once. The triggered flag is not serialized on a saved game. This means that even if a previous savegame is loaded after the node has been triggered, the node won't be triggered again.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Any</td>
<td>Input value</td>
</tr>
<tr>
<td>Reset</td>
<td>Any</td>
<td>Resets the input and allows new activation to occur</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Any</td>
<td>Receives the active input only once</td>
</tr>
</tbody>
</table>

RandomSelect node

Used to pass the activated input value to a random number of outputs.
### Logic Nodes

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Any</td>
<td>Input value</td>
</tr>
<tr>
<td>MinOut</td>
<td>Integer</td>
<td>Minimum number of outputs to trigger</td>
</tr>
<tr>
<td>MaxOut</td>
<td>Integer</td>
<td>Maximum number of outputs to trigger</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out0 - Out9</td>
<td>Any</td>
<td>Receives active input values</td>
</tr>
</tbody>
</table>

#### RandomTrigger node

Used to trigger one of the outputs in random order.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Any</td>
<td>Input value</td>
</tr>
<tr>
<td>Reset</td>
<td>Any</td>
<td>Resets the activations to 0</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out0 - Out9</td>
<td>Any</td>
<td>Output value</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when all outputs have been triggered</td>
</tr>
</tbody>
</table>
SelectCondition node

Used to trigger the output based on the state of the Condition node.

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>Condition</td>
<td>Boolean</td>
<td>Condition value</td>
</tr>
<tr>
<td>InTrue</td>
<td>Any</td>
<td>Value sent to Out when Condition is true</td>
</tr>
<tr>
<td>InFalse</td>
<td>Any</td>
<td>Value sent to Out when Condition is false</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Any</td>
<td>Output value</td>
</tr>
</tbody>
</table>

Sequencer node

Used to trigger one of the outputs in sequential order for each input activation.

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Any</td>
<td>Input value</td>
</tr>
<tr>
<td>Closed</td>
<td>Boolean</td>
<td>If true, blocks all inputs</td>
</tr>
<tr>
<td>Open</td>
<td>Any</td>
<td>Sets Closed to false</td>
</tr>
<tr>
<td>Close</td>
<td>Any</td>
<td>Sets Closed to true</td>
</tr>
<tr>
<td>Reset</td>
<td>Any</td>
<td>Forces active output to Out0</td>
</tr>
<tr>
<td>Reverse</td>
<td>Boolean</td>
<td>If true, the order of output activation is reversed</td>
</tr>
</tbody>
</table>
### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out0 - Out9</td>
<td>Any</td>
<td>Outputs are triggered in sequential order for each input activation</td>
</tr>
</tbody>
</table>

### XOR node

Used to perform a logical XOR operation on the input ports. If one of the inputs is true, the output is true. If both inputs are true or are false, the output is false.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>A</td>
<td>Boolean</td>
<td>First input</td>
</tr>
<tr>
<td>B</td>
<td>Boolean</td>
<td>Second input</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Boolean</td>
<td>If one of the inputs is true, the output is true. If both inputs are true or are false, the output is false</td>
</tr>
<tr>
<td>OnTrue</td>
<td>Boolean</td>
<td>Triggers if Out is true</td>
</tr>
<tr>
<td>OnFalse</td>
<td>Boolean</td>
<td>Triggers if Out is false</td>
</tr>
</tbody>
</table>

### Material Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard’s new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to define material settings.

**Topics**

- EntityMaterialChange node (p. 405)
- EntityMaterialParams node (p. 405)
Material Nodes

- MaterialClone node (p. 406)
- MaterialParams node (p. 406)
- SetObjectMaterial node (p. 407)

EntityMaterialChange node

Used to apply the specified material to an entity.

![EntityMaterialChange node](image)

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>MaterialName</td>
<td>String</td>
<td>Name of material to apply</td>
</tr>
<tr>
<td>Reset</td>
<td>Any</td>
<td>Reset to the original material</td>
</tr>
<tr>
<td>SerializeChanges</td>
<td>Boolean</td>
<td>Serialize the change</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td>Outputs the name of the material</td>
</tr>
</tbody>
</table>

EntityMaterialParams node

Used to get the entity's material parameters.

![EntityMaterialParams node](image)

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Slot</td>
<td>Integer</td>
<td>Material slot</td>
</tr>
<tr>
<td>SubMtlId</td>
<td>Integer</td>
<td>Submaterial ID</td>
</tr>
<tr>
<td>ParamFloat</td>
<td>String</td>
<td>Float parameter to be set</td>
</tr>
<tr>
<td>ValueFloat</td>
<td>Float</td>
<td>Sets float parameter value</td>
</tr>
<tr>
<td>ParamColor</td>
<td>String</td>
<td>Color parameter to be set</td>
</tr>
</tbody>
</table>
Material Nodes

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ValueColor</td>
<td>Vec3</td>
<td>Color value to be set</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FloatValue</td>
<td>Float</td>
<td>Current float value</td>
</tr>
<tr>
<td>ColorValue</td>
<td>Vec3</td>
<td>Current color value</td>
</tr>
</tbody>
</table>

**MaterialClone node**

Used to clone an entity's material or reset it back to the original.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Reset</td>
<td>Any</td>
<td>Resets to the original material</td>
</tr>
<tr>
<td>Slot</td>
<td>Integer</td>
<td>Material slot</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onCloned</td>
<td>Any</td>
<td>Activated when material is cloned</td>
</tr>
<tr>
<td>OnReset</td>
<td>Any</td>
<td>Activated when material is reset</td>
</tr>
</tbody>
</table>

**MaterialParams node**

Used to get the specified material's parameters.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
</tbody>
</table>

406
Lumberyard Legacy Reference

**MaterialFX Nodes**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaterialName</td>
<td>String</td>
<td>Material name</td>
</tr>
<tr>
<td>SubMtlId</td>
<td>Integer</td>
<td>Submaterial name</td>
</tr>
<tr>
<td>ParamFloat</td>
<td>String</td>
<td>Float parameter to be set</td>
</tr>
<tr>
<td>ValueFloat</td>
<td>Float</td>
<td>Value of the float parameter</td>
</tr>
<tr>
<td>ParamColor</td>
<td>String</td>
<td>Color parameter to be set</td>
</tr>
<tr>
<td>ValueColor</td>
<td>Vec3</td>
<td>Value of the color parameter</td>
</tr>
<tr>
<td>SerializeChanges</td>
<td>Boolean</td>
<td>Serialize the change</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FloatValue</td>
<td>Float</td>
<td>Current float value</td>
</tr>
<tr>
<td>ColorValue</td>
<td>Vec3</td>
<td>Current color value</td>
</tr>
</tbody>
</table>

**SetObjectMaterial node**

Used to set an object's (render node) material to the specified position.

```plaintext
Material=material1
ObjectType=object_type
Position=0,0,0
Activate=activate
```

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Material</td>
<td>String</td>
<td>Set object material</td>
</tr>
<tr>
<td>ObjectType</td>
<td>Integer</td>
<td>Object type</td>
</tr>
<tr>
<td>Position</td>
<td>Vec3</td>
<td>Position to set material at</td>
</tr>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
</tbody>
</table>

**MaterialFX Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about **Script Canvas**, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.
You can use the following flow graph nodes to define material FX settings.

**Note**
These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at Lumberyard Downloads.

**Topics**
- HUDEndFX node (p. 408)
- HUDStartFX node (p. 408)

**HUDEndFX node**
The MaterialFX end node for an HUD.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>MaterialFX end node</td>
</tr>
</tbody>
</table>

**HUDStartFX node**
The MaterialFX start node for an HUD.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Triggered automatically by the material effect</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Started</td>
<td>Any</td>
<td>Triggered when the material effect has started</td>
</tr>
<tr>
<td>Distance</td>
<td>Float</td>
<td>Outputs the distance to the player</td>
</tr>
<tr>
<td>Param1</td>
<td>Float</td>
<td>Custom parameter 1</td>
</tr>
<tr>
<td>Param2</td>
<td>Float</td>
<td>Custom parameter 2</td>
</tr>
<tr>
<td>Intensity</td>
<td>Float</td>
<td>Dynamic value set by game code</td>
</tr>
<tr>
<td>BlendOutTime</td>
<td>Float</td>
<td>Outputs the material effect blend out time in seconds</td>
</tr>
</tbody>
</table>
Math Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use these flow graph nodes to define math operations.

Topics
- Abs node (p. 410)
- Add node (p. 410)
- AnglesToDir node (p. 411)
- ArcCos node (p. 411)
- ArcSin node (p. 411)
- ArcTan node (p. 412)
- ArcTan2 node (p. 412)
- BooleanFrom node (p. 413)
- BooleanTo node (p. 413)
- Calculate node (p. 413)
- Ceil node (p. 414)
- Clamp node (p. 414)
- Cosine node (p. 415)
- Counter node (p. 415)
- DirToAngles node (p. 415)
- Div node (p. 416)
- Equal node (p. 416)
- EvenOrOdd node (p. 417)
- Floor node (p. 417)
- InRange node (p. 417)
- Less node (p. 418)
- Mod node (p. 419)
- Mul node (p. 419)
- Noise1D node (p. 419)
- Noise3D node (p. 420)
- PortCounter node (p. 420)
- Power node (p. 421)
- Random node (p. 422)
- Reciprocal node (p. 422)
- Remainder node (p. 423)
- Round node (p. 423)
Lumberyard Legacy Reference
Math Nodes

- SetColor node (p. 423)
- SetNumber node (p. 424)
- SinCos node (p. 424)
- Sine node (p. 425)
- Sqrt node (p. 425)
- Sub node (p. 425)
- Tangent node (p. 426)
- UpDownCounter node (p. 426)

### Abs node

Used to calculate the absolute value of the input.

![Abs node](image)

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>A</td>
<td>Float</td>
<td>Input</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Absolute value of the input</td>
</tr>
</tbody>
</table>

### Add node

Used to add the two input values.

![Add node](image)

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>A</td>
<td>Float</td>
<td>First operand</td>
</tr>
<tr>
<td>B</td>
<td>Float</td>
<td>Second operand</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Absolute value of the input</td>
</tr>
</tbody>
</table>
**AnglesToDir node**

Used to convert the input angle to a unit vector direction.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Angles</td>
<td>Vec3</td>
<td>Input angle</td>
</tr>
</tbody>
</table>

**ArcCos node**

Used to calculate the inverse cosine of the input.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>In</td>
<td>Float</td>
<td>Input angle</td>
</tr>
</tbody>
</table>

**ArcSin node**

Used to calculate the inverse sine of the input.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>In</td>
<td>Float</td>
<td>Input angle</td>
</tr>
</tbody>
</table>
### ArcTan node

Used to calculate the inverse tangent of the input.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>In</td>
<td>Float</td>
<td>Input angle</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Inverse tangent (Arctan) of the input</td>
</tr>
</tbody>
</table>

### ArcTan2 node

Used to calculate the inverse tangent of the two inputs.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>X</td>
<td>Float</td>
<td>X input value</td>
</tr>
<tr>
<td>Y</td>
<td>Float</td>
<td>Y input value</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Inverse tangent (Arctan) of the Y and X inputs</td>
</tr>
</tbody>
</table>
BooleanFrom node

Used to convert the Boolean input value (0 or 1) to true or false.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Boolean</td>
<td>Boolean input (0 or 1)</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>Float</td>
<td>Triggers if input is false (0)</td>
</tr>
<tr>
<td>True</td>
<td>Float</td>
<td>Triggers if input is true (1)</td>
</tr>
</tbody>
</table>

BooleanTo node

Used to convert the inputs to a Boolean 0 or 1 value.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Float</td>
<td>Will output true if event is received on this input</td>
</tr>
<tr>
<td>False</td>
<td>Float</td>
<td>Will output false if event is received on this input</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Boolean</td>
<td>Outputs true (1) or false (0) depending on input state</td>
</tr>
</tbody>
</table>

Calculate node

Used to calculate the output value based on the operation performed on the two inputs.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Port</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Operation</td>
<td>Integer</td>
<td>The mathematical operation to be performed</td>
</tr>
<tr>
<td>A</td>
<td>Float</td>
<td>First operand</td>
</tr>
<tr>
<td>B</td>
<td>Float</td>
<td>Second operand</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Result of operation on A and B</td>
</tr>
</tbody>
</table>

**Ceil node**

Used to output the ceiling value of the input.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Float</td>
<td>Input</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Ceiling input value</td>
</tr>
</tbody>
</table>

**Clamp node**

Used to clamp the output value to the Min and Max range.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Float</td>
<td>Input value</td>
</tr>
<tr>
<td>Min</td>
<td>Float</td>
<td>Minimum clamp value</td>
</tr>
<tr>
<td>Max</td>
<td>Float</td>
<td>Maximum clamp value</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Triggered if the input is clamped within the range</td>
</tr>
</tbody>
</table>
Cosine node

Used to output the cosine of the input.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>In</td>
<td>Float</td>
<td>Input in degrees</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Cosine of the input</td>
</tr>
</tbody>
</table>

Counter node

Used to output the number of times the input has been activated.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Any</td>
<td>Input</td>
</tr>
<tr>
<td>Reset</td>
<td>Any</td>
<td>Resets the counter</td>
</tr>
<tr>
<td>Max</td>
<td>Integer</td>
<td>Maximum value of the counter before it is reset</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Integer</td>
<td>Number of times that the input was activated</td>
</tr>
</tbody>
</table>

DirToAngles node

Used to convert the input vector direction to an angle.
### Math Nodes

#### Dir node

- **Inputs**
  - **Port** | **Type** | **Description**
  - Dir | Vec3 | Vector direction
  - Roll | Float | Roll input

- **Outputs**
  - **Port** | **Type** | **Description**
  - Angles | Vec3 | Converts the direction to an angle in degrees

#### Div node

Used to divide input A by input B.

- **Inputs**
  - **Port** | **Type** | **Description**
  - Activate | Any | Triggers the node
  - A | Float | Dividend input
  - B | Float | Divisor input

- **Outputs**
  - **Port** | **Type** | **Description**
  - Out | Float | Division of A by B

#### Equal node

Used to check if the two inputs are equal in value.

- **Inputs**
  - **Port** | **Type** | **Description**
  - Activate | Any | Triggers the node
  - A | Float | First operand
  - B | Float | Second operand
### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Boolean</td>
<td>True if the two inputs are equal in value</td>
</tr>
<tr>
<td>OnTrue</td>
<td>Any</td>
<td>Triggered if the inputs are equal in value</td>
</tr>
<tr>
<td>OnFalse</td>
<td>Any</td>
<td>Triggered if the inputs are not equal in value</td>
</tr>
</tbody>
</table>

### EvenOrOdd node

Used to check if the input is an even or odd value.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Integer</td>
<td>Input</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Odd</td>
<td>Any</td>
<td>Triggered if the input is an odd value</td>
</tr>
<tr>
<td>Even</td>
<td>Any</td>
<td>Triggered if the input is an even value</td>
</tr>
</tbody>
</table>

### Floor node

Used to output the floor of the input.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Float</td>
<td>Input</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Floored input</td>
</tr>
</tbody>
</table>

### InRange node

Used to check if the input is within the Min and Max value range.
## Math Nodes

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>In</td>
<td>Float</td>
<td>Input</td>
</tr>
<tr>
<td>Min</td>
<td>Float</td>
<td>Minimum value of the range</td>
</tr>
<tr>
<td>Max</td>
<td>Float</td>
<td>Maximum value of the range</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Boolean</td>
<td>True if the input is within the range</td>
</tr>
<tr>
<td>OnTrue</td>
<td>Any</td>
<td>Triggered if the input is within the range</td>
</tr>
<tr>
<td>OnFalse</td>
<td>Any</td>
<td>Triggered if the input is outside of the range</td>
</tr>
</tbody>
</table>

### Less node

Used to check whether the A input is less than the B input.

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Boolean</td>
<td>True if A is less than B</td>
</tr>
<tr>
<td>OnTrue</td>
<td>Any</td>
<td>Triggered is A is less than B</td>
</tr>
<tr>
<td>OnFalse</td>
<td>Any</td>
<td>Triggered if A is greater than B</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Boolean</td>
<td>True if A is less than B</td>
</tr>
<tr>
<td>OnTrue</td>
<td>Any</td>
<td>Triggered is A is less than B</td>
</tr>
<tr>
<td>OnFalse</td>
<td>Any</td>
<td>Triggered if A is greater than B</td>
</tr>
</tbody>
</table>
Mod node

Used to calculate the modulus of the two inputs.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>A</td>
<td>Float</td>
<td>First operand</td>
</tr>
<tr>
<td>B</td>
<td>Float</td>
<td>Second operand</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Float</td>
<td>Modulus of the two inputs</td>
</tr>
</tbody>
</table>

Mul node

Used to multiply the two inputs.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>A</td>
<td>Float</td>
<td>First operand</td>
</tr>
<tr>
<td>B</td>
<td>Float</td>
<td>Second operand</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Multiplication of the two inputs</td>
</tr>
</tbody>
</table>

Noise1D node

Used to multiply the scalar input by the frequency and amplitude.
### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Float</td>
<td>Scalar Input value to sample noise at</td>
</tr>
<tr>
<td>Frequency</td>
<td>Float</td>
<td>Frequency</td>
</tr>
<tr>
<td>Amplitude</td>
<td>Float</td>
<td>Amplitude</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Multiplication of X by Frequency and Amplitude values</td>
</tr>
</tbody>
</table>

### Noise3D node

Used to multiple the vector input by the frequency and amplitude.

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>Vec3</td>
<td>Vector input value to sample noise at</td>
</tr>
<tr>
<td>Frequency</td>
<td>Float</td>
<td>Frequency</td>
</tr>
<tr>
<td>Amplitude</td>
<td>Float</td>
<td>Amplitude</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Multiplication of V by Frequency and Amplitude values</td>
</tr>
</tbody>
</table>

### PortCounter node

Used to count the number of activated inputs.
Lumberyard Legacy Reference
Math Nodes

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>Any</td>
<td>Resets PortCount and TotalCount</td>
</tr>
<tr>
<td>PortThreshold</td>
<td>Integer</td>
<td>PortCount threshold value</td>
</tr>
<tr>
<td>TotalThreshold</td>
<td>Integer</td>
<td>TotalCount threshold value</td>
</tr>
<tr>
<td>In00 - In15</td>
<td>Any</td>
<td>Inputs</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PortCount</td>
<td>Integer</td>
<td>Number of ports that have been set</td>
</tr>
<tr>
<td>TotalCount</td>
<td>Integer</td>
<td>Sum of all times any of the input ports have been set</td>
</tr>
<tr>
<td>PortTrigger</td>
<td>Boolean</td>
<td>Triggered when PortCount reaches PortThreshold</td>
</tr>
<tr>
<td>TotalTrigger</td>
<td>Boolean</td>
<td>Triggered when TotalCount reaches TotalThreshold</td>
</tr>
</tbody>
</table>

Power node

Used to calculate the Base input raised to the Power exponent.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>Base</td>
<td>Float</td>
<td>Base input</td>
</tr>
<tr>
<td>Power</td>
<td>Float</td>
<td>Exponent input</td>
</tr>
</tbody>
</table>
### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Base input value raised to the Power exponent</td>
</tr>
</tbody>
</table>

### Random node

Used to generate a random number between the Min and Max values, both as an integer and as a floating point number.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate</td>
<td>Any</td>
<td>Generates a random number</td>
</tr>
<tr>
<td>Min</td>
<td>Float</td>
<td>Minimum value of the random number</td>
</tr>
<tr>
<td>Max</td>
<td>Float</td>
<td>Maximum value of the random number</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Output as floating-point number</td>
</tr>
<tr>
<td>OutRounded</td>
<td>Integer</td>
<td>Output rounded to next integer value</td>
</tr>
</tbody>
</table>

### Reciprocal node

Used to output the reciprocal value of the input.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>A</td>
<td>Float</td>
<td>Input</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Reciprocal of the input</td>
</tr>
</tbody>
</table>
**Remainder node**

Used to output the remainder value of A divided by B.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>A</td>
<td>Float</td>
<td>Dividend input</td>
</tr>
<tr>
<td>B</td>
<td>Float</td>
<td>Divisor input</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Remainder of the inputs</td>
</tr>
</tbody>
</table>

**Round node**

Used to round the input floating point value to an integer output.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>In</td>
<td>Float</td>
<td>Floating-point Input</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutRounded</td>
<td>Integer</td>
<td>Rounded integer value of the input</td>
</tr>
</tbody>
</table>

**SetColor node**

Used to output the input vector color when the Set input is activated.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>Any</td>
<td>Triggers input to output</td>
</tr>
</tbody>
</table>
# Lumberyard Legacy Reference
## Math Nodes

### Port

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Vec3</td>
<td>Vector input value</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Vec3</td>
<td>Input value when Set is triggered</td>
</tr>
</tbody>
</table>

### SetNumber node

Used to output the input scalar number when the Set input is activated.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>In</td>
<td>Float</td>
<td>Input</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Outputs the input</td>
</tr>
</tbody>
</table>

### SinCos node

Used to calculate the sine and cosine of the input.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>In</td>
<td>Float</td>
<td>Input angle in degrees</td>
</tr>
</tbody>
</table>
### Sine node

Used to calculate the sine of the input.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>In</td>
<td>Float</td>
<td>Input angle in degrees</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Sine of the input</td>
</tr>
</tbody>
</table>

### Sqrt node

Used to calculate the square root of the input.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>A</td>
<td>Float</td>
<td>Input</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Square root of the input</td>
</tr>
</tbody>
</table>

### Sub node

Used to subtract the two inputs.
## Math Nodes

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>In</td>
<td>Float</td>
<td>First operand</td>
</tr>
<tr>
<td>Out</td>
<td>Float</td>
<td>Second operand</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Subtraction of the two inputs</td>
</tr>
</tbody>
</table>

### Tangent node

Used to calculate the tangent of the input.

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>In</td>
<td>Float</td>
<td>Input angle in degrees</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Tangent of the input</td>
</tr>
</tbody>
</table>

### UpDownCounter node

Used to output an up or down counter.
Mission Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard’s new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use these flow graph nodes to configure mission-related settings. Game tokens are useful as variables used for passing data between flow graphs or within a flow graph, or for storing data between levels.

Topics

- GameToken node (p. 428)
- GameTokenCheck node (p. 428)
- GameTokenCheckMulti node (p. 429)
- GameTokenGet node (p. 429)
- GameTokenModify node (p. 430)
- GameTokenSet node (p. 430)
- GameTokensLevelToLevelRestore node (p. 431)
- GameTokensLevelToLevelStore node (p. 431)
• **LoadNextLevel node (p. 431)**

### GameToken node

Used to get or set a game token. This is the most important and useful of all the mission nodes as it acts like a listener for any changes on the input.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token</td>
<td>String</td>
<td>Game token to compare. Any change in this value will trigger the TokenValue output.</td>
</tr>
<tr>
<td>CompareValue</td>
<td>String</td>
<td>Value to compare the token value against</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TokenValue</td>
<td>Any</td>
<td>Value of the game token. Triggers whenever the Token input changes value.</td>
</tr>
<tr>
<td>OnTrue</td>
<td>Boolean</td>
<td>Triggered if the token value is equal to CompareValue</td>
</tr>
<tr>
<td>OnFalse</td>
<td>Boolean</td>
<td>Triggered if the token value is not equal to CompareValue</td>
</tr>
</tbody>
</table>

### GameTokenCheck node

Used to check if the value of a game token equals a value.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Token</td>
<td>String</td>
<td>Game token to check</td>
</tr>
<tr>
<td>CompareValue</td>
<td>String</td>
<td>Value to compare the token value against</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TokenValue</td>
<td>Any</td>
<td>Value of the token</td>
</tr>
<tr>
<td>Port</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Result</td>
<td>Boolean</td>
<td>True if the token value is equal to CompareValue</td>
</tr>
<tr>
<td>OnTrue</td>
<td>Any</td>
<td>Triggered if the token value is equal to CompareValue</td>
</tr>
<tr>
<td>OnFalse</td>
<td>Any</td>
<td>Triggered if the token value is not equal to CompareValue</td>
</tr>
</tbody>
</table>

**GameTokenCheckMulti node**

Used to check if a game token is equal to any value in a list.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Token</td>
<td>String</td>
<td>Game token to check</td>
</tr>
<tr>
<td>Value0 - Value7</td>
<td>String</td>
<td>Values to compare the token value with</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TokenValue</td>
<td>Any</td>
<td>Value of the token</td>
</tr>
<tr>
<td>OneTrue</td>
<td>Any</td>
<td>Triggered if the token value is equal to at least one of the input port values</td>
</tr>
<tr>
<td>AllFalse</td>
<td>Any</td>
<td>Triggered if the token value is not equal to any of the input port values</td>
</tr>
</tbody>
</table>

**GameTokenGet node**

Used to get the value of the game token.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
</tbody>
</table>
### Token

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token</td>
<td>String</td>
<td>Game token to get</td>
</tr>
</tbody>
</table>

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutValue</td>
<td>Any</td>
<td>Displays value of the game token</td>
</tr>
</tbody>
</table>

### GameTokenModify node

Used to modify the value of a game token.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Token</td>
<td>String</td>
<td>Game token to set</td>
</tr>
<tr>
<td>Operation</td>
<td>Integer</td>
<td>Operation to perform on the token</td>
</tr>
<tr>
<td>TokenType</td>
<td>Integer</td>
<td>Token type</td>
</tr>
<tr>
<td>OtherValue</td>
<td>String</td>
<td>Value to perform operation with</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Any</td>
<td>Result of the operation</td>
</tr>
</tbody>
</table>

### GameTokenSet node

Used to set the value of a game token.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Token</td>
<td>String</td>
<td>Game token to set</td>
</tr>
<tr>
<td>TokenValue</td>
<td>String</td>
<td>Value of token</td>
</tr>
</tbody>
</table>
OutValue | Any | Outputs token value

**GameTokensLevelToLevelRestore node**

Used to restore the values of all game tokens in a level that were stored in the previous level using the `GameTokensLevelToLevelStore` node.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
</tbody>
</table>

**GameTokensLevelToLevelStore node**

Used to store the values of all game tokens in a level.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Token0 - Token7</td>
<td>String</td>
<td>Stores token values</td>
</tr>
</tbody>
</table>

**LoadNextLevel node**

Used to load the next level.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>NextLevel</td>
<td>String</td>
<td>Ends the current level and loads the next level</td>
</tr>
</tbody>
</table>
Module Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

Module nodes are project-specific, user-created nodes. The nodes listed here are used in the Samples projects, which are located at `lumberyard_version\dev\SamplesProject\Levels\Samples`.

Topics
- Call_Character_Controller_Robot node (p. 432)
- Call_Character_Controller_Robot_Completed node (p. 433)
- Call_Free_Cam_Controller node (p. 434)
- Call_VR_Character_Controller_Robot node (p. 434)
- Utils:UserIDToModuleID node (p. 435)

Call_Character_Controller_Robot node

Used to call a character's controller robot.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call</td>
<td>Any</td>
<td>Calls the module</td>
</tr>
<tr>
<td>Instanced</td>
<td>Boolean</td>
<td>Whether the module is instanced or not.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Any</td>
<td>Cancels the module</td>
</tr>
<tr>
<td>InstanceID</td>
<td>Integer</td>
<td>Instance ID</td>
</tr>
<tr>
<td>Controller_Ref_Box_Pitch</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>Controller_Ref_Box_Move</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>Awesome_Sphere_X</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>Awesome_Sphere_Y</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>Awesome_Sphere_Z</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>Robot_Heading</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>Camera_Fov</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>Robot_Fovy</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>Head_Tilt_Parent_Y</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>Head_Tilt_Parent_X</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>Head_Tilt_Parent_Z</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
## Module Nodes

### Port

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Awesome_Sphere</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>Robot_Head</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>Camera_Rig</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>Robot_Body</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>Head_Tilt_Parent</td>
<td>Integer</td>
<td>Integer</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnCalled</td>
<td>Integer</td>
<td>Triggers when module is started</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Successful status</td>
</tr>
<tr>
<td>Cancelled</td>
<td>Any</td>
<td>Failed status</td>
</tr>
</tbody>
</table>

### Call_Character_Controller_Robot_Completed node

Used to call a character's controller robot.

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call</td>
<td>Any</td>
<td>Calls the module</td>
</tr>
<tr>
<td>Instanced</td>
<td>Boolean</td>
<td>Whether the module is instanced or not.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Any</td>
<td>Cancels the module</td>
</tr>
<tr>
<td>InstanceID</td>
<td>Integer</td>
<td>Instance ID</td>
</tr>
<tr>
<td>Controller_Ref_Box_Pitch</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>Controller_Ref_Box_Move</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>Awesome_Sphere</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>Robot_Head</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>Camera_Rig</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>Robot_Body</td>
<td>Integer</td>
<td>Integer</td>
</tr>
</tbody>
</table>
### Call_Free_Cam_Controller node

Used to call a camera controller.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call</td>
<td>Any</td>
<td>Calls the module</td>
</tr>
<tr>
<td>Instanced</td>
<td>Boolean</td>
<td>Whether the module is instanced or not.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Any</td>
<td>Cancels the module</td>
</tr>
<tr>
<td>InstanceID</td>
<td>Integer</td>
<td>Instance ID</td>
</tr>
<tr>
<td>Entity_ID_Camera</td>
<td>Integer</td>
<td>Integer</td>
</tr>
</tbody>
</table>
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call</td>
<td>Any</td>
<td>Calls the module</td>
</tr>
<tr>
<td>Instanced</td>
<td>Boolean</td>
<td>Whether the module is instanced or not.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Any</td>
<td>Cancels the module</td>
</tr>
<tr>
<td>InstanceID</td>
<td>Integer</td>
<td>Instance ID</td>
</tr>
<tr>
<td>Sphere</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>Ref_Move</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>Camera</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>Ref_Camera_Yaw</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>Camera_Fulcrum</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>HMD_Fulcrum</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>Head_Tilt_Parent</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>Robot_Body</td>
<td>Integer</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnCalled</td>
<td>Integer</td>
<td>Triggers when module is started</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Successful status</td>
</tr>
<tr>
<td>Cancelled</td>
<td>Any</td>
<td>Failed status</td>
</tr>
</tbody>
</table>

**Utils:UserIDToModuleID node**

Used to map a user ID to a module instance ID.
Lumberyard Legacy Reference
Movement Nodes

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get</td>
<td>Any</td>
<td>Gets the module instance ID for the user ID</td>
</tr>
<tr>
<td>Set</td>
<td>Any</td>
<td>Gets the module instance ID for the user ID</td>
</tr>
<tr>
<td>UserID</td>
<td>Integer</td>
<td>User ID</td>
</tr>
<tr>
<td>ModuleID</td>
<td>Integer</td>
<td>Module instance ID</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModuleID</td>
<td>Integer</td>
<td>Module instance ID for the user ID</td>
</tr>
</tbody>
</table>

Movement Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard’s new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to specify entity movements.

Topics

- MoveEntityTo node (p. 436)
- RotateEntity node (p. 437)
- RotateEntityTo node (p. 438)

MoveEntityTo node

Used to move an entity to a destination position at a defined speed or in a defined interval of time.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination</td>
<td>Vec3</td>
<td>Position of the destination.</td>
</tr>
<tr>
<td>DynamicUpdate</td>
<td>Boolean</td>
<td>Indicates if destination position is to be followed if it changes.</td>
</tr>
<tr>
<td>ValueType</td>
<td>Integer</td>
<td>Type of input: Speed, Time,</td>
</tr>
<tr>
<td>Value</td>
<td>Float</td>
<td>Speed (m/sec) or Time (sec) value</td>
</tr>
<tr>
<td>EaseInDistance</td>
<td>Float</td>
<td>Distance from destination at which the entity starts slowing down</td>
</tr>
<tr>
<td>EaseOutDistance</td>
<td>Float</td>
<td>Distance from destination at which the entity starts speeding up</td>
</tr>
<tr>
<td>CoordSys</td>
<td>Integer</td>
<td>Coordinate system of the destination: Parent, World, or Local.</td>
</tr>
<tr>
<td>Start</td>
<td>Any</td>
<td>Starts movement</td>
</tr>
<tr>
<td>Stop</td>
<td>Any</td>
<td>Stops movement</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current</td>
<td>Vec3</td>
<td>Current position</td>
</tr>
<tr>
<td>OnStart</td>
<td>Any</td>
<td>Activated when Start is triggered</td>
</tr>
<tr>
<td>OnStop</td>
<td>Any</td>
<td>Activated when Stop is triggered</td>
</tr>
<tr>
<td>Finish</td>
<td>Any</td>
<td>Activated when destination is reached</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Activated when destination is reached or Stop is triggered.</td>
</tr>
</tbody>
</table>

**RotateEntity node**

Used to rotate an entity at a defined speed.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables updates</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables updates</td>
</tr>
<tr>
<td>Velocity</td>
<td>Vec3</td>
<td>Angular velocity (degrees/sec)</td>
</tr>
<tr>
<td>Port</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>CoordSys</td>
<td>Integer</td>
<td>Coordinate system for rotation: World, Local</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CurrentDegrees</td>
<td>Vec3</td>
<td>Current rotation in degrees</td>
</tr>
<tr>
<td>CurrentRadians</td>
<td>Vec3</td>
<td>Current rotation in radians</td>
</tr>
</tbody>
</table>

**RotateEntityTo node**

Used to rotate an entity at a defined speed or in a defined interval of time.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination</td>
<td>Vec3</td>
<td>Destination position (in degrees)</td>
</tr>
<tr>
<td>DynamicUpdate</td>
<td>Boolean</td>
<td>If dynamic updates are enabled or not</td>
</tr>
<tr>
<td>ValueType</td>
<td>Integer</td>
<td>Type of input value: Speed (m/sec) or Time (sec)</td>
</tr>
<tr>
<td>Value</td>
<td>Float</td>
<td>Value of Speed or Time</td>
</tr>
<tr>
<td>CoordSys</td>
<td>Integer</td>
<td>Coordinate system of the destination: Parent, World, Local</td>
</tr>
<tr>
<td>Start</td>
<td>Any</td>
<td>Starts movement</td>
</tr>
<tr>
<td>Stop</td>
<td>Any</td>
<td>Stops movement</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CurrentDeg</td>
<td>Vec3</td>
<td>Current rotation in degrees</td>
</tr>
<tr>
<td>CurrentRad</td>
<td>Vec3</td>
<td>Current rotation in radians</td>
</tr>
<tr>
<td>OnStart</td>
<td>Any</td>
<td>Activated when Start input is triggered</td>
</tr>
<tr>
<td>OnStop</td>
<td>Any</td>
<td>Activated when Stop input is triggered</td>
</tr>
<tr>
<td>Finish</td>
<td>Any</td>
<td>Activated when destination rotation is reached</td>
</tr>
</tbody>
</table>
Lumberyard Legacy Reference

Physics Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to configure physics.

**Topics**
- ActionImpulse node (p. 439)
- CameraProxy node (p. 440)
- CollisionListener node (p. 440)
- Constraint node (p. 441)
- Dynamics node (p. 442)
- PhysicsEnable node (p. 443)
- PhysicsSleepQuery node (p. 443)
- RayCast node (p. 444)
- RaycastCamera node (p. 444)

**ActionImpulse node**

Used to apply an impulse to an entity.

```
ActionImpulse
```

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Impulse</td>
<td>Vec3</td>
<td>Impulse vector</td>
</tr>
<tr>
<td>AngularImpulse</td>
<td>Vec3</td>
<td>Angular impulse vector</td>
</tr>
<tr>
<td>Point</td>
<td>Vec3</td>
<td>Location impulse is applied at</td>
</tr>
</tbody>
</table>
### CameraProxy node

Used to create a entity camera proxy.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>Any</td>
<td>Creates a physicalized camera proxy if one does not exist</td>
</tr>
<tr>
<td>EntityHost</td>
<td>Any</td>
<td>Syncs proxy rotation with the current view camera</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntityCamera</td>
<td>Integer</td>
<td>Retrieves the camera proxy</td>
</tr>
</tbody>
</table>

### CollisionListener node

Used to setup physics collision listeners.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddListener</td>
<td>Any</td>
<td>Adds collision listener</td>
</tr>
<tr>
<td>IgnoreSameNode</td>
<td>Boolean</td>
<td>Suppresses events if both colliders are registered via the same node</td>
</tr>
<tr>
<td>RemoveListener</td>
<td>Any</td>
<td>Removes collision listener</td>
</tr>
</tbody>
</table>
## Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IdA</td>
<td>Any</td>
<td>ID of the first colliding entity</td>
</tr>
<tr>
<td>PartIdA</td>
<td>Integer</td>
<td>Part ID inside the first colliding entity</td>
</tr>
<tr>
<td>IdB</td>
<td>Any</td>
<td>ID of the second colliding entity</td>
</tr>
<tr>
<td>PartIdB</td>
<td>Integer</td>
<td>Part ID inside the second colliding entity</td>
</tr>
<tr>
<td>Point</td>
<td>Vec3</td>
<td>Location of collision point</td>
</tr>
<tr>
<td>Normal</td>
<td>Vec3</td>
<td>Collision normal</td>
</tr>
<tr>
<td>SurfacetypeA</td>
<td>String</td>
<td>Surface type of the first colliding entity</td>
</tr>
<tr>
<td>SurfacetypeB</td>
<td>String</td>
<td>Surface type of the second colliding entity</td>
</tr>
<tr>
<td>HitImpulse</td>
<td>Float</td>
<td>Collision impulse along the normal</td>
</tr>
</tbody>
</table>

## Constraint node

Used to create a physics constraint.

```
PhysicsConstraint
  Create
  Break
  Id=1000
  EntityA=1
  EntityB=1
  PartIdA=1
  PartIdB=1
  Point=0,0,0
  IgnoreCollisions=1
  MaxLinear=1
  MaxTorque=1
  MaxTorqueRelative=1
  MaxTorqueRelative=1
  MaxTorqueRelative=1
```

## Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>Any</td>
<td>Creates the constraint</td>
</tr>
<tr>
<td>Break</td>
<td>Any</td>
<td>Breaks the constraint</td>
</tr>
<tr>
<td>Id</td>
<td>Integer</td>
<td>Constraint ID</td>
</tr>
<tr>
<td>EntityA</td>
<td>Any</td>
<td>Constraint owner entity</td>
</tr>
<tr>
<td>PartIdA</td>
<td>Integer</td>
<td>Part ID to attach to</td>
</tr>
<tr>
<td>EntityB</td>
<td>Any</td>
<td>Constraint buddy entity</td>
</tr>
<tr>
<td>PartIdB</td>
<td>Integer</td>
<td>Part ID to attach to</td>
</tr>
<tr>
<td>Point</td>
<td>Vec3</td>
<td>Connection point in worldspace</td>
</tr>
</tbody>
</table>
## Physics Nodes

### Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IgnoreCollisions</td>
<td>Boolean</td>
<td>Disables collisions between constrained entities</td>
</tr>
<tr>
<td>Breakable</td>
<td>Boolean</td>
<td>Break if force limit is reached</td>
</tr>
<tr>
<td>ForceAwake</td>
<td>Boolean</td>
<td>Make entity B always awake; restores previous sleep parameters</td>
</tr>
<tr>
<td>MaxForce</td>
<td>Float</td>
<td>Force limit</td>
</tr>
<tr>
<td>MaxTorque</td>
<td>Float</td>
<td>Rotational force (torque) force limit</td>
</tr>
<tr>
<td>MaxForceRelative</td>
<td>Any</td>
<td>Make limits relative to entity B's mass</td>
</tr>
<tr>
<td>TwistAxis</td>
<td>Boolean</td>
<td>Main rotation axis in worldspace</td>
</tr>
<tr>
<td>MinTwist</td>
<td>Float</td>
<td>Lower rotation limit around TwistAxis</td>
</tr>
<tr>
<td>MaxTwist</td>
<td>Float</td>
<td>Upper rotation limit around TwistAxis</td>
</tr>
<tr>
<td>MaxBend</td>
<td>Float</td>
<td>Maximum bend of the TwistAxis</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Integer</td>
<td>Constraint ID</td>
</tr>
<tr>
<td>Broken</td>
<td>Boolean</td>
<td>Triggered when the constraint breaks</td>
</tr>
</tbody>
</table>

### Dynamics node

Used to output the dynamic state of an entity.

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enables updates</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables updates</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Velocity</td>
<td>Vec3</td>
<td>Velocity of entity</td>
</tr>
<tr>
<td>Acceleration</td>
<td>Vec3</td>
<td>Acceleration of entity</td>
</tr>
<tr>
<td>AngularVelocity</td>
<td>Vec3</td>
<td>Angular velocity of entity</td>
</tr>
<tr>
<td>Port</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>AngularAcceleration</td>
<td>Vec3</td>
<td>Angular acceleration of entity</td>
</tr>
<tr>
<td>Mass</td>
<td>Float</td>
<td>Mass of entity</td>
</tr>
</tbody>
</table>

### PhysicsEnable node

Used to enable and disable physics.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnablePhysics</td>
<td>Any</td>
<td>Enables physics for entity</td>
</tr>
<tr>
<td>DisablePhysics</td>
<td>Any</td>
<td>Disables physics for entity</td>
</tr>
<tr>
<td>EnableAI</td>
<td>Any</td>
<td>Enables AI for entity</td>
</tr>
<tr>
<td>DisableAI</td>
<td>Any</td>
<td>Disables AI for entity</td>
</tr>
</tbody>
</table>

### PhysicsSleepQuery node

Used to return the sleeping state of the physics of a given entity.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition</td>
<td>Boolean</td>
<td>Sleeping state of the entity</td>
</tr>
<tr>
<td>Reset</td>
<td>Boolean</td>
<td>Resets the node</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sleeping</td>
<td>Boolean</td>
<td>Sleeping state of the entity</td>
</tr>
<tr>
<td>OnSleep</td>
<td>Any</td>
<td>Triggered when the entity physics switches to sleep</td>
</tr>
<tr>
<td>OneAwake</td>
<td>Any</td>
<td>Triggered when the entity physics switches to awake</td>
</tr>
</tbody>
</table>
**RayCast node**

Used to perform a raycast relative to an entity.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Direction</td>
<td>Vec3</td>
<td>Direction of the raycast</td>
</tr>
<tr>
<td>MaxLength</td>
<td>Float</td>
<td>Maximum length of the raycast</td>
</tr>
<tr>
<td>Position</td>
<td>Vec3</td>
<td>Ray start position relative to the entity</td>
</tr>
<tr>
<td>TransformDirection</td>
<td>Boolean</td>
<td>Transforms direction by entity orientation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoHit</td>
<td>Any</td>
<td>Triggered if no object was hit by the raycast</td>
</tr>
<tr>
<td>Hit</td>
<td>Any</td>
<td>Triggered if an object was hit by the raycast</td>
</tr>
<tr>
<td>RayDirection</td>
<td>Vec3</td>
<td>Direction of the cast ray</td>
</tr>
<tr>
<td>HitDistance</td>
<td>Float</td>
<td>Distance to the hit object</td>
</tr>
<tr>
<td>HitPoint</td>
<td>Vec3</td>
<td>Position of the hit</td>
</tr>
<tr>
<td>HitNormal</td>
<td>Vec3</td>
<td>Normal of the surface at the HitPoint</td>
</tr>
<tr>
<td>HitSurfaceType</td>
<td>Integer</td>
<td>Surface type index of the surface hit</td>
</tr>
<tr>
<td>HitEntity</td>
<td>Any</td>
<td>ID of the entity that was hit</td>
</tr>
</tbody>
</table>

**RaycastCamera node**

Used to perform a raycast relative to a camera.
### Prefab Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about [Script Canvas](https://docs.aws.amazon.com,lumberyard), Lumberyard's new visual scripting environment.

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 about the latest features, see the [Amazon Lumberyard User Guide](https://docs.aws.amazon.com,lumberyard).

You can use the following flow graph nodes to configure prefab settings.

**Topics**
- EventSource node (p. 445)

**EventSource node**

Used to add an event source inside of a prefab for it to be handled like an instance.
Lumberyard Legacy Reference
ProceduralMaterial Nodes

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrefabName</td>
<td>String</td>
<td>Name of the prefab</td>
</tr>
<tr>
<td>InstanceName</td>
<td>String</td>
<td>Name of the prefab instance</td>
</tr>
<tr>
<td>EventName</td>
<td>String</td>
<td>Name of the event associated with the prefab</td>
</tr>
<tr>
<td>FireEvent</td>
<td>Any</td>
<td>Fires the associated event</td>
</tr>
<tr>
<td>EventId</td>
<td>Integer</td>
<td>ID of the event</td>
</tr>
<tr>
<td>EventIndex</td>
<td>Integer</td>
<td>Position of the event in the index</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventFired</td>
<td>Any</td>
<td>Triggered when the event has fired</td>
</tr>
</tbody>
</table>

ProceduralMaterial Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to configure procedural material settings.

Topics
- GetGraphInstanceId node (p. 447)
- GetInputFloat node (p. 447)
- GetInputFloat2 node (p. 448)
- GetInputFloat3 node (p. 448)
- GetInputFloat4 node (p. 449)
- GetInput node (p. 449)
- GetInput2 node (p. 450)
- GetInput3 node (p. 450)
- GetInput4 node (p. 451)
• QueueGraphInstance node (p. 452)
• RenderASync node (p. 452)
• RenderSync node (p. 452)
• SetInputFloat node (p. 453)
• SetInputFloat2 node (p. 453)
• SetInputFloat3 node (p. 454)
• SetInputFloat4 node (p. 454)
• SetInputImage node (p. 455)
• SetInputInt node (p. 455)
• SetInputInt2 node (p. 456)
• SetInputInt3 node (p. 456)
• SetInputInt4 node (p. 457)

GetGraphInstanceID node

Used to get the graph instance ID.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProceduralMaterial</td>
<td>String</td>
<td>Name of the procedural material</td>
</tr>
<tr>
<td>GraphicIndex</td>
<td>Integer</td>
<td>Graph index</td>
</tr>
<tr>
<td>Get</td>
<td>Any</td>
<td>Get the graph index</td>
</tr>
</tbody>
</table>

GetInputFloat node

Used to get the Substance input floating point value.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphInstanceID</td>
<td>Integer</td>
<td>Graph instance ID</td>
</tr>
<tr>
<td>ParameterName</td>
<td>String</td>
<td>Parameter name</td>
</tr>
</tbody>
</table>
Port | Type | Description
---|---|---
Get | Any | Get parameter value

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value1</td>
<td>Float</td>
<td>Outputs parameter value</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when input completes</td>
</tr>
</tbody>
</table>

**GetInputFloat2 node**

Used to get the Substance input floating point values.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphInstanceID</td>
<td>Integer</td>
<td>Graph instance ID</td>
</tr>
<tr>
<td>ParameterName</td>
<td>String</td>
<td>Parameter name</td>
</tr>
<tr>
<td>Get</td>
<td>Any</td>
<td>Get parameter value</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value1</td>
<td>Integer</td>
<td>Outputs parameter value 1</td>
</tr>
<tr>
<td>Value2</td>
<td>Integer</td>
<td>Outputs parameter value 2</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when input completes</td>
</tr>
</tbody>
</table>

**GetInputFloat3 node**

Used to get the Substance input floating point values.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphInstanceID</td>
<td>Integer</td>
<td>Graph instance ID</td>
</tr>
<tr>
<td>ParameterName</td>
<td>String</td>
<td>Parameter name</td>
</tr>
<tr>
<td>Port</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Get</td>
<td>Any</td>
<td>Get parameter value</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value1</td>
<td>Float</td>
<td>Outputs parameter value 1</td>
</tr>
<tr>
<td>Value2</td>
<td>Float</td>
<td>Outputs parameter value 2</td>
</tr>
<tr>
<td>Value3</td>
<td>Float</td>
<td>Outputs parameter value 3</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when input completes</td>
</tr>
</tbody>
</table>

### GetInputFloat4 node

Used to get the Substance input floating point values.

![GetInputFloat4 node diagram](image)

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphInstanceID</td>
<td>Integer</td>
<td>Graph instance ID</td>
</tr>
<tr>
<td>ParameterName</td>
<td>String</td>
<td>Parameter name</td>
</tr>
<tr>
<td>Get</td>
<td>Any</td>
<td>Get parameter value</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value1</td>
<td>Float</td>
<td>Outputs parameter value 1</td>
</tr>
<tr>
<td>Value2</td>
<td>Float</td>
<td>Outputs parameter value 2</td>
</tr>
<tr>
<td>Value3</td>
<td>Float</td>
<td>Outputs parameter value 3</td>
</tr>
<tr>
<td>Value4</td>
<td>Float</td>
<td>Outputs parameter value 4</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when input completes</td>
</tr>
</tbody>
</table>

### GetInput node

Used to get the Substance input value.

![GetInput node diagram](image)
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphInstanceID</td>
<td>Integer</td>
<td>Graph instance ID</td>
</tr>
<tr>
<td>ParameterName</td>
<td>String</td>
<td>Parameter name</td>
</tr>
<tr>
<td>Get</td>
<td>Any</td>
<td>Get parameter value</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value1</td>
<td>Float</td>
<td>Outputs parameter value</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when input completes</td>
</tr>
</tbody>
</table>

GetInput2 node

Used to get the Substance input value.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphInstanceID</td>
<td>Integer</td>
<td>Graph instance ID</td>
</tr>
<tr>
<td>ParameterName</td>
<td>String</td>
<td>Parameter name</td>
</tr>
<tr>
<td>Get</td>
<td>Any</td>
<td>Get parameter value</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value1</td>
<td>Float</td>
<td>Outputs parameter value 1</td>
</tr>
<tr>
<td>Value2</td>
<td>Float</td>
<td>Outputs parameter value 2</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when input completes</td>
</tr>
</tbody>
</table>

GetInput3 node

Used to get the Substance input value.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphInstanceID</td>
<td>Integer</td>
<td>Graph instance ID</td>
</tr>
<tr>
<td>ParameterName</td>
<td>String</td>
<td>Parameter name</td>
</tr>
<tr>
<td>Get</td>
<td>Any</td>
<td>Get parameter value</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value1</td>
<td>Float</td>
<td>Outputs parameter value 1</td>
</tr>
<tr>
<td>Value2</td>
<td>Float</td>
<td>Outputs parameter value 2</td>
</tr>
<tr>
<td>Value3</td>
<td>Float</td>
<td>Outputs parameter value 3</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when input completes</td>
</tr>
</tbody>
</table>

GetInput4 node

Used to get the Substance input value.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphInstanceID</td>
<td>Integer</td>
<td>Graph instance ID</td>
</tr>
<tr>
<td>ParameterName</td>
<td>String</td>
<td>Parameter name</td>
</tr>
<tr>
<td>Get</td>
<td>Any</td>
<td>Get parameter value</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value1</td>
<td>Float</td>
<td>Outputs parameter value 1</td>
</tr>
<tr>
<td>Value2</td>
<td>Float</td>
<td>Outputs parameter value 2</td>
</tr>
<tr>
<td>Value3</td>
<td>Float</td>
<td>Outputs parameter value 3</td>
</tr>
<tr>
<td>Value4</td>
<td>Float</td>
<td>Outputs parameter value 4</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when input completes</td>
</tr>
</tbody>
</table>
QueueGraphInstance node

Used to queue to graph instance.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphInstanceID</td>
<td>Integer</td>
<td>Graph instance ID</td>
</tr>
<tr>
<td>Add</td>
<td>Any</td>
<td>Add graph instance ID to the queue</td>
</tr>
</tbody>
</table>

RenderASync node

Used to render queued graphs asynchronously.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Render</td>
<td>Any</td>
<td>Begin rendering graph instance asynchronously</td>
</tr>
</tbody>
</table>

RenderSync node

Used to render queued graphs synchronously.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Render</td>
<td>Any</td>
<td>Begin rendering graph instance synchronously</td>
</tr>
<tr>
<td>RenderBegin</td>
<td>Any</td>
<td>Triggered when rendering has started</td>
</tr>
<tr>
<td>RenderComplete</td>
<td>Any</td>
<td>Triggered when rendering has completed</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RenderComplete</td>
<td>Any</td>
<td>Triggered when rendering has completed</td>
</tr>
</tbody>
</table>

**SetInputFloat node**

Used to set the Substance input floating point value.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphInstanceID</td>
<td>Integer</td>
<td>Graph instance ID</td>
</tr>
<tr>
<td>ParameterName</td>
<td>String</td>
<td>Parameter name</td>
</tr>
<tr>
<td>Value1</td>
<td>Float</td>
<td>Floating point parameter value to set</td>
</tr>
<tr>
<td>Apply</td>
<td>Any</td>
<td>Set parameter value</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when input completes</td>
</tr>
</tbody>
</table>

**SetInputFloat2 node**

Used to set the Substance input floating point values.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphInstanceID</td>
<td>Integer</td>
<td>Graph instance ID</td>
</tr>
<tr>
<td>ParameterName</td>
<td>String</td>
<td>Parameter name</td>
</tr>
<tr>
<td>Value1</td>
<td>Float</td>
<td>Floating point parameter value 1 to set</td>
</tr>
<tr>
<td>Value2</td>
<td>Float</td>
<td>Floating point parameter value 2 to set</td>
</tr>
<tr>
<td>Apply</td>
<td>Any</td>
<td>Set parameter value</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when input completes</td>
</tr>
</tbody>
</table>

**SetInputFloat3 node**

Used to set the Substance input floating point values.

![Diagram of SetInputFloat3 node]

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphInstanceID</td>
<td>Integer</td>
<td>Graph instance ID</td>
</tr>
<tr>
<td>ParameterName</td>
<td>String</td>
<td>Parameter name</td>
</tr>
<tr>
<td>Value1</td>
<td>Float</td>
<td>Floating point parameter value 1 to set</td>
</tr>
<tr>
<td>Value2</td>
<td>Float</td>
<td>Floating point parameter value 2 to set</td>
</tr>
<tr>
<td>Value3</td>
<td>Float</td>
<td>Floating point parameter value 3 to set</td>
</tr>
<tr>
<td>Apply</td>
<td>Any</td>
<td>Set parameter value</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when input completes</td>
</tr>
</tbody>
</table>

**SetInputFloat4 node**

Used to set the Substance input floating point values.

![Diagram of SetInputFloat4 node]

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphInstanceID</td>
<td>Integer</td>
<td>Graph instance ID</td>
</tr>
<tr>
<td>ParameterName</td>
<td>String</td>
<td>Parameter name</td>
</tr>
<tr>
<td>Value1</td>
<td>Float</td>
<td>Floating point parameter value 1 to set</td>
</tr>
<tr>
<td>Value2</td>
<td>Float</td>
<td>Floating point parameter value 2 to set</td>
</tr>
<tr>
<td>Value3</td>
<td>Float</td>
<td>Floating point parameter value 3 to set</td>
</tr>
<tr>
<td>Apply</td>
<td>Any</td>
<td>Set parameter value</td>
</tr>
</tbody>
</table>
Lumberyard Legacy Reference
ProceduralMaterial Nodes

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value2</td>
<td>Float</td>
<td>Floating point parameter value 2 to set</td>
</tr>
<tr>
<td>Value3</td>
<td>Float</td>
<td>Floating point parameter value 3 to set</td>
</tr>
<tr>
<td>Value4</td>
<td>Float</td>
<td>Floating point parameter value 4 to set</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apply</td>
<td>Any</td>
<td>Set parameter value</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when input completes</td>
</tr>
</tbody>
</table>

### SetInputImage node

Used to set the Substance input image.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphInstanceID</td>
<td>Integer</td>
<td>Graph instance ID</td>
</tr>
<tr>
<td>ParameterName</td>
<td>String</td>
<td>Parameter name</td>
</tr>
<tr>
<td>Texture</td>
<td>String</td>
<td>Image to be set</td>
</tr>
<tr>
<td>Apply</td>
<td>Any</td>
<td>Set input image</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when input completes</td>
</tr>
</tbody>
</table>

### SetInputInt node

Used to set the Substance input value.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphInstanceID</td>
<td>Integer</td>
<td>Graph instance ID</td>
</tr>
</tbody>
</table>

455
### SetInputInt2 node

Used to set the Substance input values.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphInstanceID</td>
<td>Integer</td>
<td>Graph instance ID</td>
</tr>
<tr>
<td>ParameterName</td>
<td>String</td>
<td>Parameter name</td>
</tr>
<tr>
<td>Value1</td>
<td>Float</td>
<td>Parameter value 1 to set</td>
</tr>
<tr>
<td>Value2</td>
<td>Float</td>
<td>Parameter value 2 to set</td>
</tr>
<tr>
<td>Apply</td>
<td>Any</td>
<td>Set parameter value</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when input completes</td>
</tr>
</tbody>
</table>

### SetInputInt3 node

Used to set the Substance input values.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphInstanceID</td>
<td>Integer</td>
<td>Graph instance ID</td>
</tr>
<tr>
<td>ParameterName</td>
<td>String</td>
<td>Parameter name</td>
</tr>
<tr>
<td>Value1</td>
<td>Float</td>
<td>Parameter value 1 to set</td>
</tr>
<tr>
<td>Value2</td>
<td>Float</td>
<td>Parameter value 2 to set</td>
</tr>
<tr>
<td>Value3</td>
<td>Float</td>
<td>Parameter value 3 to set</td>
</tr>
<tr>
<td>Apply</td>
<td>Any</td>
<td>Set parameter value</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when input completes</td>
</tr>
</tbody>
</table>
Lumberyard Legacy Reference  
ProceduralMaterial Nodes

## Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphInstanceID</td>
<td>Integer</td>
<td>Graph instance ID</td>
</tr>
<tr>
<td>ParameterName</td>
<td>String</td>
<td>Parameter name</td>
</tr>
<tr>
<td>Value1</td>
<td>Float</td>
<td>Parameter value 1 to set</td>
</tr>
<tr>
<td>Value2</td>
<td>Float</td>
<td>Parameter value 2 to set</td>
</tr>
<tr>
<td>Value3</td>
<td>Float</td>
<td>Parameter value 3 to set</td>
</tr>
<tr>
<td>Apply</td>
<td>Any</td>
<td>Set parameter value</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when input completes</td>
</tr>
</tbody>
</table>

## SetInputInt4 node

Used to set the Substance input values.

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphInstanceID</td>
<td>Integer</td>
<td>Graph instance ID</td>
</tr>
<tr>
<td>ParameterName</td>
<td>String</td>
<td>Parameter name</td>
</tr>
<tr>
<td>Value1</td>
<td>Float</td>
<td>Parameter value 1 to set</td>
</tr>
<tr>
<td>Value2</td>
<td>Float</td>
<td>Parameter value 2 to set</td>
</tr>
<tr>
<td>Value3</td>
<td>Float</td>
<td>Parameter value 3 to set</td>
</tr>
<tr>
<td>Value4</td>
<td>Float</td>
<td>Parameter value 4 to set</td>
</tr>
<tr>
<td>Apply</td>
<td>Any</td>
<td>Set parameter value</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when input completes</td>
</tr>
</tbody>
</table>
Stereo Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to configure stereographic settings.

Topics
- ReadStereoParameters node (p. 458)
- StereoParameters node (p. 458)

ReadStereoParameters node

Used to read the HUD stereo display parameters.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Any</td>
<td>Start reading stereo values</td>
</tr>
<tr>
<td>Stop</td>
<td>Any</td>
<td>Stop reading stereo values</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EyeDistance</td>
<td>Float</td>
<td>Outputs eye distance</td>
</tr>
<tr>
<td>ScreenDistance</td>
<td>Float</td>
<td>Outputs screen distance</td>
</tr>
<tr>
<td>HUDDistance</td>
<td>Float</td>
<td>Outputs HUD distance</td>
</tr>
<tr>
<td>Flipped</td>
<td>Boolean</td>
<td>Output if stereo is flipped</td>
</tr>
</tbody>
</table>

StereoParameters node

Used to output the HUD stereo display parameters.
### String Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about [Script Canvas](https://docs.aws.amazon.com/lumberyard/latest/scriptcanvas/index.html), Lumberyard's new visual scripting environment.

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](https://docs.aws.amazon.com/lumberyard/latest/scriptcanvas/index.html) or the command line. To learn about the latest features, see the [Amazon Lumberyard User Guide](https://docs.aws.amazon.com/lumberyard/latest/scriptcanvas/index.html).

You can use the following flow graph nodes to configure strings.

**Topics**
- Collect node (p. 459)
- Compare node (p. 460)
- Concat node (p. 460)
- ReplaceString node (p. 461)
- SetString node (p. 461)
- Split node (p. 462)
- URLDecode node (p. 462)

**Collect node**

Used to collect a string.
Lumberyard Legacy Reference
String Nodes

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Collects the strings and triggers the output</td>
</tr>
<tr>
<td>Input</td>
<td>String</td>
<td>Each string that will be joined</td>
</tr>
<tr>
<td>JoinString</td>
<td>String</td>
<td>String to use between all collected strings</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CollectedString</td>
<td>String</td>
<td>Outputs the collected string set</td>
</tr>
</tbody>
</table>

Compare node

Used to compare two strings.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare</td>
<td>Any</td>
<td>Triggers string comparison</td>
</tr>
<tr>
<td>A</td>
<td>String</td>
<td>First string to compare</td>
</tr>
<tr>
<td>B</td>
<td>String</td>
<td>Second string to compare</td>
</tr>
<tr>
<td>IgnoreCase</td>
<td>Boolean</td>
<td>Ignores casing</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Integer</td>
<td>Outputs -1 if string A less than string B, 0 if String A equals string B, 1 if string A is greater than string B</td>
</tr>
<tr>
<td>False</td>
<td>Any</td>
<td>Triggers if string A does not equal string B</td>
</tr>
<tr>
<td>True</td>
<td>Any</td>
<td>Triggers if string A equals string B</td>
</tr>
</tbody>
</table>

Concat node

Used to concatenate two strings.
### String Nodes

#### String concatenation node

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>Any</td>
<td>Triggers string concatenation</td>
</tr>
<tr>
<td>String1</td>
<td>String</td>
<td>First string to concatenate</td>
</tr>
<tr>
<td>String2</td>
<td>String</td>
<td>Second string to concatenate</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>String</td>
<td>Outputs the concatenated string</td>
</tr>
</tbody>
</table>

#### ReplaceString node

Used to replace a string.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>String</td>
<td>Triggers string replacement</td>
</tr>
<tr>
<td>Replace</td>
<td>String</td>
<td>The string to replace</td>
</tr>
<tr>
<td>ReplaceWith</td>
<td>String</td>
<td>The new string to replace with</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutString</td>
<td>String</td>
<td>Outputs the replaced string</td>
</tr>
</tbody>
</table>

#### SetString node

Used to set a string value.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>Any</td>
<td>Sends the string to the output</td>
</tr>
<tr>
<td>Port</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>---------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>In</td>
<td>String</td>
<td>String to set on</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>String</td>
<td>Outputs the string value</td>
</tr>
</tbody>
</table>

**Split node**

Used to split a string.

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the string split</td>
</tr>
<tr>
<td>Input</td>
<td>String</td>
<td>The string to split</td>
</tr>
<tr>
<td>Separator</td>
<td>String</td>
<td>Character to separate the string on. If you pass a string, only the first character will be used</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Split0 - Split4</td>
<td>String</td>
<td>Outputs the specific string split</td>
</tr>
</tbody>
</table>

**URLDecode node**

Used to decode the URL of a string.

### Input

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>String</td>
<td>String to URL decode</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DecodedString</td>
<td>String</td>
<td>Outputs the URL-decoded string</td>
</tr>
</tbody>
</table>
System Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to configure system settings.

Topics
- Container:Create node (p. 463)
- Container:Edit node (p. 463)
- Container:Iterate node (p. 464)

Container:Create node

Used to create a container.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Integer</td>
<td>Container ID</td>
</tr>
<tr>
<td>Create</td>
<td>Any</td>
<td>Creates a container</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>Integer</td>
<td>Triggers when an error occurs</td>
</tr>
<tr>
<td>Success</td>
<td>Any</td>
<td>Tiggers when a container is created</td>
</tr>
<tr>
<td>Id</td>
<td>Integer</td>
<td>Outputs the container ID</td>
</tr>
</tbody>
</table>

Container:Edit node

Used to edit a container.
### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Integer</td>
<td>Container ID</td>
</tr>
<tr>
<td>Add</td>
<td>Any</td>
<td>Adds the passed item to the container</td>
</tr>
<tr>
<td>AddUnique</td>
<td>Any</td>
<td>Adds the passed item if it didn't exist</td>
</tr>
<tr>
<td>Remove</td>
<td>Any</td>
<td>Removes all occurrences of the current item</td>
</tr>
<tr>
<td>Clear</td>
<td>Any</td>
<td>Empties the container</td>
</tr>
<tr>
<td>GetCount</td>
<td>Any</td>
<td>Gets the number of items in the container</td>
</tr>
<tr>
<td>Delete</td>
<td>Any</td>
<td>Deletes the container</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>Integer</td>
<td>Triggers when an error occurs</td>
</tr>
<tr>
<td>Success</td>
<td>Any</td>
<td>Triggers when the operation successfully completed</td>
</tr>
</tbody>
</table>

### Container: Iterate node

Used to iterate over a container.

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Integer</td>
<td>Container ID</td>
</tr>
<tr>
<td>Start</td>
<td>Any</td>
<td>Starts iterating the container</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>Integer</td>
<td>Triggers when an error occurs</td>
</tr>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggers when the operation successfully completed</td>
</tr>
<tr>
<td>Out</td>
<td>Any</td>
<td>Outputs the container ID</td>
</tr>
</tbody>
</table>
Time Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use these flow graph nodes to define time settings.

Topics
- Delay (p. 465)
- FrameDelay (p. 466)
- MeasureTime (p. 466)
- RandomDelay (p. 467)
- RealTime (p. 467)
- ServerTime (p. 468)
- Time (p. 469)
- TimeOfDay (p. 469)
- TimeOfDayLoadDefinitionFile (p. 470)
- TimeOfDayTransitionTrigger (p. 471)
- TimeOfDayTrigger (p. 472)
- TimedCounter (p. 473)
- Timer (p. 473)

Delay

Delays passing the signal from [In] to [Out] for the specified length of time (seconds).

![Delay Node](image)

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Any</td>
<td>Value to pass after the specified delay time</td>
</tr>
<tr>
<td>Delay</td>
<td>Float</td>
<td>Delay time in seconds</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values: 0 – 100</td>
</tr>
<tr>
<td>resetOnInput</td>
<td>Boolean</td>
<td>When set to true, resets the node with each input, setting the delay counter to 0 and erasing previous inputs</td>
</tr>
<tr>
<td>Port</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values: 0=false</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>Any</td>
<td>Value that is passed after the specified frame delay</td>
</tr>
</tbody>
</table>

**FrameDelay**

Delays passing the signal from [In] to [Out] for the specified number of frames.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Any</td>
<td>Value to pass after the specified delay time</td>
</tr>
<tr>
<td>NFrames</td>
<td>Integer</td>
<td>Number of frames to delay passing the signal from [In] to [Out]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values: 0 – 100</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out</td>
<td>Any</td>
<td>Value that is passed after the specified frame delay</td>
</tr>
</tbody>
</table>

**MeasureTime**

Measures the elapsed time.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Trigger to begin measuring time as it passes</td>
</tr>
<tr>
<td>Port</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>Stop</td>
<td>Any</td>
<td>Trigger to stop measuring the elapsed time</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Started</td>
<td>Any</td>
<td>Triggered on start</td>
</tr>
<tr>
<td>Stopped</td>
<td>Any</td>
<td>Triggered on stop</td>
</tr>
<tr>
<td>Elapsed</td>
<td>Any</td>
<td>Elapsed time in seconds</td>
</tr>
</tbody>
</table>

**RandomDelay**

Delays passing the signal from [In] to [Out] for a random amount of time (seconds) within the [MinDelay, MaxDelay] interval.

```
Time:RandomDelay
├── In → Out
├── MinDelay = 1
├── MaxDelay = 2
```

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Any</td>
<td>Value to pass after the specified delay time</td>
</tr>
<tr>
<td>MinDelay</td>
<td>Float</td>
<td>Minimum random delay time in seconds</td>
</tr>
<tr>
<td>MaxDelay</td>
<td>Float</td>
<td>Maximum random delay time in seconds</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Any</td>
<td>Value that is passed after the specified delay time</td>
</tr>
</tbody>
</table>

**RealTime**

Reads your system time. RealTime can be used to display time on screen (such as a player's watch) or synchronize the time of day with real world time.
## Time Nodes

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>force_update</td>
<td>Any</td>
<td>Forces an update of the system time</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours</td>
<td>Integer</td>
<td>Current hour as reported by your system time</td>
</tr>
<tr>
<td>Minutes</td>
<td>Integer</td>
<td>Current minutes as reported by your system time</td>
</tr>
<tr>
<td>Seconds</td>
<td>Integer</td>
<td>Current seconds as reported by your system time</td>
</tr>
<tr>
<td>Datetime</td>
<td>String</td>
<td>Outputs your system date and time</td>
</tr>
<tr>
<td>Epoch</td>
<td>Integer</td>
<td>Current epoch as reported by your system time</td>
</tr>
</tbody>
</table>

### ServerTime

Reads the server time and reports the current time (seconds or milliseconds) for the specified period.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basetime</td>
<td>Float</td>
<td>Base time in seconds. The server time output is relative to the base time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values: 0 – 100</td>
</tr>
<tr>
<td>Period</td>
<td>Float</td>
<td>Number of seconds that should pass before the timer resets to 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values: 0 – 100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secs</td>
<td>Integer</td>
<td>Current time in seconds, relative to the base time</td>
</tr>
<tr>
<td>Port</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Msecs</td>
<td>Integer</td>
<td>Current time in milliseconds, relative to the base time</td>
</tr>
<tr>
<td>Period</td>
<td>Boolean</td>
<td>Triggers the Period output once for each period of time, as specified by the Period input. Valid values: 0=false</td>
</tr>
</tbody>
</table>

**Time**

Outputs the total number of seconds from the start of the game, ticking once per frame.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paused</td>
<td>Boolean</td>
<td>Pauses the time output when set to true. Default value: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values: 0=false</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>seconds</td>
<td>Float</td>
<td>Current time in seconds</td>
</tr>
<tr>
<td>tick</td>
<td>Any</td>
<td>Triggers a tick once per frame</td>
</tr>
</tbody>
</table>

**TimeOfDay**

Changes the speed at which the time of day progresses and reads the current TimeOfDay setting.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Float</td>
<td>Time of day in hours</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 0</td>
</tr>
</tbody>
</table>
## Time Nodes

### Port | Type | Description
--- | --- | ---
SetTime | Any | Trigger to change the time of day to the value specified for the Time parameter
ForceUpdate | Boolean | Immediately updates the sky when set to `true`. Default value: 0 Valid values: 0=false | 1=true
GetTime | Any | Retriggers the CurTime output without updating the value of the output
Speed | Float | Sets the speed at which the time of day changes. Default value: 1 Valid values: 0 – 100
SetSpeed | Any | Trigger to change the time of day speed to the value specified for the Speed parameter
GetSpeed | Any | Retriggers the CurTime output without updating the value of the output

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| CurTime | float | Current time of day based on when the Set input was last triggered. Use the Get input to retrigger this output and keep the current value for the output
| CurSpeed | float | Speed for the current time of day based on when the SetSpeed input was last triggered. Use the GetSpeed input to retrigger this output and keep the current value for the output

### TimeOfDayLoadDefinitionFile

Loads a Time of Day (TOD) definition file.

`TimeOfDayLoadDefinitionFile` | Load | Filename |
--- | --- | ---
Success | Fail |

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Load | Any | Trigger to load and read the Time of Day definition file
### Time Nodes

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filename</td>
<td>String</td>
<td>Name of the XML file to load and read. The file must be in the level directory</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Triggered when the Time of Day definition file has successfully loaded</td>
</tr>
<tr>
<td>Fail</td>
<td>Any</td>
<td>Triggered if the Time of Day definition file was not successfully loaded</td>
</tr>
</tbody>
</table>

#### TimeOfDayTransitionTrigger

Triggers sun position transitions when a specific time of day is reached.

![Image of TimeOfDayTransitionTrigger node]

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Float</td>
<td>Total length of time to blend the level's current time to the specified time. Set this value to −1 to disable time of day blending</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values: 0 – 24</td>
</tr>
<tr>
<td>Duration</td>
<td>Float</td>
<td>Blend duration in seconds.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values: 0 – 100</td>
</tr>
<tr>
<td>SunLatitude</td>
<td>Float</td>
<td>Blends the level's current sun latitude value to the specified latitude in degrees. Set this value to −1 to disable latitude blending.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: −1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values: 0 – 100</td>
</tr>
<tr>
<td>SunLongitude</td>
<td>Float</td>
<td>Blends the level's current sun longitude value to the specified latitude in degrees. Set this value to −1 to disable longitude blending.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Time Nodes

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SunPositionUpdateInterval</td>
<td>Float</td>
<td>Amount of time in seconds between updates to reposition the sun. Set this value to 0 seconds to constantly update the sun position during the transition. Default value: 1 Valid values: 0 – 100</td>
</tr>
<tr>
<td>ForceUpdateInterval</td>
<td>Float</td>
<td>Amount of time in seconds between updates to the time of day. Set this value to 0 seconds to constantly update the time of day during the transition. Default value: 1 Valid values: 0 – 100</td>
</tr>
<tr>
<td>Start</td>
<td>Any</td>
<td>Starts the transition.</td>
</tr>
<tr>
<td>Pause</td>
<td>Any</td>
<td>Pauses or resumes the transition</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Triggered when the transition is finished</td>
</tr>
</tbody>
</table>

#### TimeOfDayTrigger

Triggers an action when a specific time of day is reached.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Boolean</td>
<td>Set this value to true to enable the trigger Default value: 1 Valid values: 0=false</td>
</tr>
<tr>
<td>Time</td>
<td>Float</td>
<td>Triggers the action at the specified time of day Default value: 0 Valid values: 0 – 100</td>
</tr>
</tbody>
</table>
**Lumberyard Legacy Reference**

**Time Nodes**

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Float</td>
<td>Displays the current value for TimeOfDay. Triggered when the specified time of day has been reached</td>
</tr>
</tbody>
</table>

### TimedCounter

Counts the number of ticks. Starting from 0, the counter increments by 1 every time the amount of time specified for the Period input has passed. When the counter reaches the value specified for the Limit input, the Finished output is triggered.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Trigger to start the counter. If it is already running, this resets the counter.</td>
</tr>
<tr>
<td>Stop</td>
<td>Any</td>
<td>Stops the counter</td>
</tr>
<tr>
<td>Continue</td>
<td>Any</td>
<td>Resumes the counter</td>
</tr>
<tr>
<td>Period</td>
<td>Float</td>
<td>Tick period in seconds</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values: 0 – 100</td>
</tr>
<tr>
<td>Limit</td>
<td>Integer</td>
<td>Default value: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values: 0 – 100</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finished</td>
<td>Any</td>
<td>Trigger indicating the counter is finished. The value that was provided as the Start input is the same as the Finished value.</td>
</tr>
<tr>
<td>Count</td>
<td>Integer</td>
<td>Value for the tick counter</td>
</tr>
</tbody>
</table>

### Timer

Outputs the count from minimum to maximum, ticking for the specified period.
Twitch Nodes

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>period</td>
<td>Float</td>
<td>Tick period in seconds</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values: 0 – 100</td>
</tr>
<tr>
<td>min</td>
<td>Integer</td>
<td>Minimum value for the timer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values: 0 – 100</td>
</tr>
<tr>
<td>max</td>
<td>Integer</td>
<td>Maximum value for the timer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values: 0 – 100</td>
</tr>
<tr>
<td>paused</td>
<td>Boolean</td>
<td>Pauses the timer when set to true.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Valid values: 0=false</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Integer</td>
<td>Total count for the specified period</td>
</tr>
</tbody>
</table>

Twitch Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use these flow graph nodes to configure Twitch-related settings.

You the attached entity dynamically by using the Choose Entity input port that is included on a number of flow graph nodes.
Twitch ChatPlay General Nodes

You can use the following flow graph nodes to configure general Twitch ChatPlay-related settings.

**Twitch:Chatplay:Available node**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Void</td>
<td>Checks the availability of Twitch ChatPlay.</td>
</tr>
</tbody>
</table>

**Twitch:Chatplay:Channel node**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel</td>
<td>String</td>
<td>Twitch channel name</td>
</tr>
<tr>
<td>Connect</td>
<td>Void</td>
<td>Initiates connection; idempotent if called while already connected or connecting. Resets the Error output state</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Void</td>
<td>Initiates disconnection; idempotent if called while already disconnected or disconnecting</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connected</td>
<td>Boolean</td>
<td>Current state of the connection to the channel</td>
</tr>
</tbody>
</table>
### Lumberyard Legacy Reference

#### Twitch Nodes

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connecting</td>
<td>Boolean</td>
<td>Indicates whether the node is currently attempting to connect</td>
</tr>
<tr>
<td>Error</td>
<td>Boolean</td>
<td>Indicates an error has occurred</td>
</tr>
</tbody>
</table>

**Twitch:Chatplay:DisconnectAll node**

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisconnectAll</td>
<td>Void</td>
<td>Disconnects all Twitch ChatPlay channels</td>
</tr>
</tbody>
</table>

**Twitch:Chatplay:Keyword node**

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel</td>
<td>String</td>
<td>Twitch channel name</td>
</tr>
<tr>
<td>Keyword</td>
<td>String</td>
<td>Keyword to match</td>
</tr>
<tr>
<td>Start</td>
<td>Void</td>
<td>Starts scanning for keywords; idempotent if called while already started</td>
</tr>
<tr>
<td>Stop</td>
<td>Void</td>
<td>Stops scanning for a keywords; idempotent if called while already stopped</td>
</tr>
<tr>
<td>Reset</td>
<td>Integer</td>
<td>Controls the initial signal count; changes to Reset are applied immediately to the current signal count</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal</td>
<td>Integer</td>
<td>Event that fires when the keyword is received on the specified channel; the value is incremented by +1 each time a keyword is received</td>
</tr>
<tr>
<td>Active</td>
<td>Boolean</td>
<td>Indicates whether the node is currently active; true if signals can occur (set as soon as Start is triggered); otherwise, false (set as soon as Stop is triggered)</td>
</tr>
<tr>
<td>Error</td>
<td>Boolean</td>
<td>Indicates that an error has occurred</td>
</tr>
</tbody>
</table>
Twitch:Chatplay:RegisterCredentials node

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Void</td>
<td>Registers the user name and OAuth token credential pair</td>
</tr>
<tr>
<td>Username</td>
<td>String</td>
<td>Twitch user name</td>
</tr>
<tr>
<td>OAuth_Token</td>
<td>String</td>
<td>OAuth tokens are generated with the Twitch Chat OAuth Password Generator.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Void</td>
<td>Signaled when done registering credentials</td>
</tr>
<tr>
<td>Error</td>
<td>Boolean</td>
<td>Indicates that an error has occurred</td>
</tr>
</tbody>
</table>

Twitch:Chatplay:UnregisterCredentials node

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Void</td>
<td>Unregisters the user name and associated OAuth token</td>
</tr>
<tr>
<td>Username</td>
<td>String</td>
<td>Twitch user name</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Void</td>
<td>Indicates when the unregistering of the credential has finished</td>
</tr>
<tr>
<td>Error</td>
<td>Boolean</td>
<td>Indicates that an error has occurred</td>
</tr>
</tbody>
</table>

Twitch:Chatplay:UnregisterAllCredentials node
**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Void</td>
<td>Unregisters all credentials at once</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Void</td>
<td>Indicates when the unregistering of all credential has finished</td>
</tr>
<tr>
<td>Error</td>
<td>Boolean</td>
<td>Indicates when an error occurs</td>
</tr>
</tbody>
</table>

**Twitch:Chatplay:Whisper node**

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Void</td>
<td>Sends the message as a whisper on behalf of the sender to the recipient</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender</td>
<td>String</td>
<td>Twitch user name of sender; must have credentials registered to successfully send a whisper (see Twitch:ChatPlay:RegisterCredentials node)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recipient</td>
<td>String</td>
<td>Twitch user name of recipient</td>
</tr>
<tr>
<td>Message</td>
<td>String</td>
<td>Message to whisper to recipient</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Void</td>
<td>Signaled if the whisper is sent successfully.</td>
</tr>
<tr>
<td>Error</td>
<td>Boolean</td>
<td>Signaled as true if an error occurred.</td>
</tr>
</tbody>
</table>

**Twitch ChatPlay Voting Nodes**

Twitch ChatPlay voting functionality make it easier to set up polls, surveys, and votes. The following figure shows an example of how various flow graph voting nodes work together.
Twitch:Chatplay:Voting:HighScores node

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Void</td>
<td>Queries the high scores</td>
</tr>
<tr>
<td>VoteName</td>
<td>String</td>
<td>The name of the vote</td>
</tr>
<tr>
<td>Reset</td>
<td>Void</td>
<td>Resets all counts to zero</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Void</td>
<td>Indicates when the operation is complete</td>
</tr>
<tr>
<td>Error</td>
<td>Void</td>
<td>Indicates that an error occurred</td>
</tr>
<tr>
<td>Count1 - Count4</td>
<td>Integer</td>
<td>Indicates the vote count for option 1, 2, 3, and 4</td>
</tr>
<tr>
<td>Name1 - Name4</td>
<td>String</td>
<td>The names for options 1, 2, 3, and 4</td>
</tr>
</tbody>
</table>
Twitch:Chatplay:Voting:Option node

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VoteName</td>
<td>String</td>
<td>The name of the vote</td>
</tr>
<tr>
<td>OptionName</td>
<td>String</td>
<td>The name of the voting option</td>
</tr>
<tr>
<td>Enable</td>
<td>Void</td>
<td>Enables the option and that it can be voted on</td>
</tr>
<tr>
<td>Disable</td>
<td>Void</td>
<td>Disables the ability to vote on the option</td>
</tr>
<tr>
<td>Remove</td>
<td>Void</td>
<td>Deletes the option</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Void</td>
<td>Indicates when the operation is complete</td>
</tr>
<tr>
<td>Error</td>
<td>Void</td>
<td>Indicates that an error occurred</td>
</tr>
</tbody>
</table>

Twitch:Chatplay:Voting:Score node

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Void</td>
<td>Queries the score for an option.</td>
</tr>
<tr>
<td>VoteName</td>
<td>String</td>
<td>The name of the vote</td>
</tr>
<tr>
<td>OptionName</td>
<td>String</td>
<td>The name of the voting option</td>
</tr>
<tr>
<td>Reset</td>
<td>Void</td>
<td>Resets the count to zero</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Void</td>
<td>Indicates when the operation is complete</td>
</tr>
<tr>
<td>Error</td>
<td>Void</td>
<td>Indicates that an error occurred</td>
</tr>
<tr>
<td>Count</td>
<td>Integer</td>
<td>Indicates the current vote count</td>
</tr>
</tbody>
</table>
Lumberyard Legacy Reference
Twitch Nodes

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Boolean</td>
<td>Indicates the current option state</td>
</tr>
</tbody>
</table>

**Twitch:Chatplay:Voting:Vote node**

- **VoteName**
  - Type: String
  - Description: The name of the vote

- **Channel**
  - Type: String
  - Description: The Twitch ChatPlay channel used to connect the vote to

- **Enable**
  - Type: Void
  - Description: Enables the vote and that it can be voted on

- **Disable**
  - Type: Void
  - Description: Disables the ability to vote on the vote

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Void</td>
<td>Indicates when the operation is complete</td>
</tr>
<tr>
<td>Error</td>
<td>Void</td>
<td>Indicates that an error occurred</td>
</tr>
</tbody>
</table>

**Twitch JoinIn Nodes**

Twitch JoinIn nodes are used to create a link that includes all the multiplayer session information necessary for other players to connect to the session.

**Twitch:Joinin:CreateLink node**

- **Activate**
  - Type: Void
  - Description: Generates a game: protocol link that allows players to join the current game

- **Command**
  - Type: String
  - Description: The commands to pass when a game launches

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>String</td>
<td>Signaled with the generated link.</td>
</tr>
</tbody>
</table>
**TwitchAPI Nodes**

TwitchAPI nodes are used to make calls to Twitch's REST API from within Lumberyard.

### Twitch:API:GET node

![Twitch:API:GET node diagram](image)

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel</td>
<td>String</td>
<td>Twitch channel name</td>
</tr>
<tr>
<td>API_Key</td>
<td>String enum</td>
<td>API call type and key; call types based on channel ID: channel, chat, follows, streams, subscriptions, and user</td>
</tr>
<tr>
<td>Get</td>
<td>Any</td>
<td>Caching has not been implemented, triggering the Get port will always start a new API call</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Any</td>
<td>Returned value for the given API call type and key; triggered whenever an API call is completed</td>
</tr>
<tr>
<td>Error</td>
<td>Integer</td>
<td>Indicates whether an error has occurred; it may be triggered with one of the following values: 1: the value for the requested API key was null 2: the value for the requested API key was of an unexpected type 3: the HTTP request failed</td>
</tr>
</tbody>
</table>

---

**UI Flow Graph Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.
You can use flow graph nodes to control the game's user interface. For example, you could specify an action that loads a specific UI canvas or set parameters for when to keep a canvas loaded.

Lumberyard features two sets of UI flow graph nodes: UIe and UI. The improved UIe nodes supersede the original (and now legacy) UI flow graph nodes. For best results when creating new flow graph nodes, use the UIe flow graph node set.

For more information on flow graphs, see Flow Graph System (p. 218).

**Topics**
- UIe Flow Graph Nodes (p. 483)
- UI Flow Graph Nodes (p. 560)
- Associating Canvases with UI Flow Graph Nodes (p. 642)
- Loading Canvases in the Flow Graph Editor (p. 644)

**UIe Flow Graph Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

The UIe flow graph node set supersedes the original UI flow graph node set (now legacy). The UIe node set behaves the same as the original UI node set, but simplifies how the nodes associate with UI canvases and UI elements.

You associate each UIe node with a UI canvas by setting the node's Choose Entity input. For information on how to associate UI canvases with UI flow graph nodes, see Associating Canvases with UI Flow Graph Nodes (p. 642).

**Topics**
- UIe Canvas Nodes (p. 483)
- UIe Component Nodes (p. 487)
- UIe Animation Node (p. 559)

**UIe Canvas Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.
You can use these flow graph nodes to perform actions on a UI canvas.

**UI:Canvas:ActionListener Node**

Listens for the specified action on a UI canvas.

**Node Inputs**

**Activate**

Initiates listening for the specified action.

**ActionName**

Name of the action to listen for.

**Node Outputs**

**OnAction**

Triggers when the canvas sends the action.

**ElementName**

Name of the UI element that triggered the action.

**ElementId**

The element Id of the fader element. Used if **ElementName** is empty.

**UI:Canvas:LoadIntoEntity Node**

Loads the specified UI canvas.

**Node Inputs**

**Activate**

Loads the canvas.

**Disabled**

Sets whether canvas is disabled initially. If disabled, the canvas is not updated or rendered.

**Node Outputs**

**OnLoad**

Sends a signal when the canvas is loaded.

**UI:Canvas:UnloadFromEntity Node**

Unloads the specified canvas.

**Node Inputs**

**Activate**

Unloads the canvas.
Node Output

Done

Sends a signal when the node's action is finished.

UI:Canvas:GetKeepLoaded Node

Gets the Boolean value of whether the canvas stays loaded when a level is unloaded.

Node Inputs

Activate

Gets whether the canvas stays loaded when the level is unloaded.

Node Output

KeepLoaded

The Boolean value of whether the canvas stays loaded if the level is unloaded. True if the canvas should stay loaded during level unload; otherwise, false.

UI:Canvas:SetKeepLoaded Node

Determines whether the canvas stays loaded when a level is unloaded.

Node Inputs

Activate

Sets whether the canvas stays loaded when the level is unloaded.

KeepLoaded

If true, causes the canvas to stay loaded when the level is unloaded.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Canvas:GetDrawOrder Node

Gets the integer draw order value for a UI canvas with respect to other UI canvases.

Node Inputs

Activate

Gets the draw order for the canvas.

Node Output

DrawOrder

Order in which the canvas draws. Higher numbers appear before lower numbers.
UIe:Canvas:SetDrawOrder Node
Sets the draw order for a UI canvas with respect to other UI canvases.

**Node Inputs**
- **Activate**
  - Sets the draw order for the canvas.
- **DrawOrder**
  - Order in which to display the canvas. Higher numbers appear before lower numbers.

**Node Output**
- **Done**
  - Sends a signal when the node's action is finished.

UIe:Canvas:GetIsPixelAligned Node
Gets the Boolean value of whether the canvas is pixel-aligned.

**Node Inputs**
- **Activate**
  - Gets whether visual element's vertices should snap to the nearest pixel.

**Node Output**
- **IsPixelAligned**
  - Boolean value. True if the visual element's vertices should snap to the nearest pixel; otherwise, false.

UIe:Canvas:SetIsPixelAligned Node
Sets whether visual element's vertices should snap to the nearest pixel.

**Node Inputs**
- **Activate**
  - Sets the pixel-aligned property for the canvas ID.
- **IsPixelAligned**
  - Boolean value that represents whether a visual element's vertices should snap to the nearest pixel.

**Node Output**
- **Done**
  - Sends a signal when the node's action is finished.

UIe:Canvas:GetEnabled Node
Gets the Boolean enabled flag of the canvas. Enabled canvases are updated and each frame rendered.
Node Inputs

Activate

Gets the enabled flag of the canvas.

Node Output

Enabled

The enabled flag of the canvas. True if enabled; otherwise, false.

UIe:Canvas:SetEnabled Node

Sets whether the canvas is enabled. Enabled canvases are updated and each frame rendered.

Node Inputs

Activate

Sets the enabled flag of the canvas.

Enabled

True if the canvas should be enabled; otherwise, false.

Node Output

Done

Sends a signal when the node's action is finished.

UIe Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

The UIe component flow graph node set supersedes the original UI component flow graph node set (now legacy). The UIe node set behaves the same as the original UI node set, but simplifies how the nodes associate with UI canvases and UI elements.

You can use these flow graph nodes to perform actions on UI elements through their components.
Each **UIe** component node has an input called **ElementName**. This input represents the name of the UI element in the UI Editor. To edit the **ElementName** input, click the `<` button (right of the text field) to automatically enter the name of the element that is currently selected in the UI Editor. Click the `.=` button to launch the UI Editor.

**Topics**

- UIe Button Component Nodes (p. 488)
- UIe Checkbox Component Nodes (p. 489)
- UIe DynamicLayout Component Nodes (p. 494)
- UIe DynamicScrollbox Component Nodes (p. 494)
- UIe Element Node (p. 495)
- UIe Fader Component Nodes (p. 499)
- UIe Image Component Nodes (p. 501)
- UIe Interactable Component Nodes (p. 504)
- UIe Layout Column Component Nodes (p. 505)
- UIe Layout Grid Component Nodes (p. 508)
- UIe Layout Row Component Nodes (p. 513)
- UIe Mask Component Nodes (p. 516)
- UIe ScrollBox Component Nodes (p. 519)
- UIe ScrollBar Component Nodes (p. 530)
- UIe Slider Component Nodes (p. 535)
- UIe Text Component Nodes (p. 542)
- UIe Text Input Component Nodes (p. 547)
- UIe Transform Component Nodes (p. 555)

### UIe Button Component Nodes

**Flow Graph** is deprecated and will be removed in a future version of Lumberyard. Learn about [Script Canvas](https://docs.aws.amazon.com/lumberyard/latest/ug/script-canvas.html), Lumberyard's new visual scripting environment.

This topic references tools and features that are **legacy**. If you want to use legacy tools in Lumberyard Editor, disable the [CryEntity Removal gem](https://docs.aws.amazon.com/lumberyard/latest/ug/cry-entity-removal-gem.html) using the [Project Configurator](https://docs.aws.amazon.com/lumberyard/latest/ug/project-configurator.html) or the command line. To learn about the latest features, see the [Amazon Lumberyard User Guide](https://docs.amazon.cn/lumberyard/latest/ug/).

Use the following flow graph nodes to perform actions on the button component.

#### UIe:Button:GetActionName Node

Gets the action name string that is emitted when the button is released.

**Node Inputs**

**Activate**

- Updates the output.

**ElementName**

- Name of the button element.
Node Output

Action

The action name associated with the button.

Ule:Button:SetActionName Node

Sets the action name string that's emitted when the button is released.

Node Inputs

Activate

Assigns the action name.

ElementName

Name of the button element.

Action

The action name string to assign to the button.

Node Output

Done

Sends a signal when the node's action is finished.

Ule Checkbox Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

Use the following flow graph nodes to perform actions on the check box component.

Ule:Checkbox:GetState Node

Gets the Boolean state of the check box.

Node Inputs

Activate

Gets the state of the check box.

ElementName

Name of the check box element.
Node Output

State

Outputs the current Boolean state of the check box.

Ule:Checkbox:SetState Node

Sets the Boolean state of the check box.

Node Inputs

Activate

Sets the state of the check box.

ElementName

Name of the check box element.

State

The Boolean state of the check box.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Checkbox:GetChangedActionName Node

Gets the action triggered when the check box value changed.

Node Inputs

Activate

Gets the changed action name.

ElementName

Name of the check box element.

Node Output

ChangedAction

The action name string value emitted when the check box value changes.

Ule:Checkbox:SetChangedActionName Node

Sets the action triggered when the check box value changed.

Node Inputs

Activate

Gets the changed action name.
**ElementName**
Name of the check box element.

**ChangedAction**
The action name string value emitted when the check box value changes.

**Ule:Checkbox:GetOptionalCheckedEntity Node**
Gets the child element to show when the check box is in the on state.

**Node Inputs**

**Activate**
Updates the output.

**ElementName**
Name of the check box element.

**Node Output**

**CheckedElement**
The child element to show when the check box is selected (in the on state).

**Ule:Checkbox:SetOptionalCheckedEntity Node**
Sets the child element to show when the check box is selected (in the on state).

**Node Inputs**

**Activate**
Updates the output.

**ElementName**
Name of the check box element.

**CheckedElement**
The child element to show when the check box is selected (in the on state).

**Node Output**

**Done**
Sends a signal when the node's action is finished.

**Ule:Checkbox:GetOptionalUncheckedEntity Node**
Gets the child element to show when the check box is deselected (in the off state).

**Node Inputs**

**Activate**
Updates the output.
ElementName

Name of the check box element.

Node Output

UncheckedElement

The child element to show when the check box is deselected (off state).

Ulle:Checkbox:SetOptionalUncheckedEntity Node

Sets the child element to show when the check box is deselected (in the off state).

Node Inputs

Activate

Updates the output.

ElementName

Name of the check box element.

UncheckedElement

The child element to show when the check box is deselected (in the off state).

Node Output

Done

Sends a signal when the node's action is finished.

Ulle:Checkbox:GetTurnOnActionName Node

Gets the action triggered when the check box is selected.

Node Inputs

Activate

Updates the output.

ElementName

Name of the check box element.

Node Output

TurnOnAction

The action name emitted when the check box is selected (turned on).

Ulle:Checkbox:SetTurnOnActionName Node

Sets the action triggered when the check box is selected (turned on).
**Node Inputs**

**Activate**  
Assigns `TurnOnAction` as the action name that is emitted when the check box is selected.

**ElementName**  
Name of the check box element.

**TurnOnAction**  
The action name emitted when the check box is selected.

**Node Output**

**Done**  
Sends a signal when the node's action is finished.

**Ule:Checkbox:GetTurnOffActionName Node**
Gets the action triggered when the check box is deselected (turned off).

**Node Inputs**

**Activate**
Update the output.

**ElementName**
Name of the check box element.

**Node Output**

**TurnOffAction**
The action name emitted when the check box is deselected.

**Ule:Checkbox:SetTurnOffActionName Node**
Sets the action triggered when the check box is deselected (turned off).

**Node Inputs**

**Activate**
Assigns `TurnOffAction` as the action name that is emitted when the check box is deselected.

**ElementName**
Name of the check box element.

**TurnOffAction**
The action name emitted when the check box is deselected.

**Node Output**

**Done**
Sends a signal when the node's action is finished.
**Ule DynamicLayout Component Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about [Script Canvas](https://lumberyard.dev), Lumberyard's new visual scripting environment.

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](https://lumberyard.dev) or the command line. To learn about the latest features, see the [Amazon Lumberyard User Guide](https://lumberyard.dev).

Use the following flow graph nodes to perform actions on the dynamic layout component.

**Ule:DynamicLayout:SetNumChildElements Node**

Sets the number of child elements of the layout element. The child elements are cloned from a prototype element.

**Node Inputs**

**Activate**

Sets the number of child elements.

**ElementName**

Name of the layout element.

**ElementId**

The element Id of the layout element. Used if `ElementName` is empty.

**NumChildElements**

The number of child elements.

**Node Output**

**Done**

Sends a signal when the node's action is finished.

**Ule DynamicScrollbox Component Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about [Script Canvas](https://lumberyard.dev), Lumberyard's new visual scripting environment.

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](https://lumberyard.dev) or the command line. To learn about the latest features, see the [Amazon Lumberyard User Guide](https://lumberyard.dev).

Use the following flow graph nodes to perform actions on the dynamic scroll box component.
**UIe:DynamicScrollBox:GetLocationIndexOfChild Node**

Gets the index of the specified child of the dynamic scroll box's content element. Use this node rather than `UIe:Element:GetIndexOfChild`, since dynamic scroll boxes create only the minimum number of children for display.

**Node Inputs**

**Activate**

Updates the output.

**ElementName**

Name of the dynamic scroll box element.

**ElementId**

The element Id of the dynamic scroll box element. Used if ElementName is empty.

**ChildElementId**

The element Id of the child.

**Node Output**

**Index**

The index of the child.

**UIe:DynamicScrollBox:RefreshContent Node**

Refreshes the dynamic scroll box by retrieving the number of the content element's children, setting up the content element according to the number of children, and notifying listeners of the child elements that are visible.

**Node Inputs**

**Activate**

Refreshes the content.

**ElementName**

Name of the dynamic scroll box element.

**ElementId**

The element Id of the dynamic scroll box element. Used if ElementName is empty.

**Node Output**

**Done**

Sends a signal when the node's action is finished.

**UIe Element Node**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.
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 about the latest features, see the Amazon Lumberyard User Guide.

Use the following flow graph nodes to perform actions on an element.

**UIe:Element:isEnabled Node**

Gets whether the element is enabled.

**Node Inputs**

- **Activate**
  - Updates the outputs.
- **CanvasID**
  - Unique identifier of the element’s canvas.
- **ElementID**
  - The element Id. Used if ElementName is empty

**Node Output**

- **State**
  - The enabled state of the element.

**UIe:Element:SetIsEnabled Node**

Sets the Boolean enabled state of the element. If an element is not enabled, neither it nor any of its children are drawn or interactive.

**Node Inputs**

- **Activate**
  - Sets the enabled state to the value of the State input.
- **ElementName**
  - Name of the element.
- **ElementID**
  - The element Id. Used if ElementName is empty
- **State**
  - The Boolean enabled state of the element.

**Node Output**

- **Done**
  - Sends a signal when the node’s action is finished.
UIe:Element:GetChildAtIndex Node

Gets the child of an element at the specified index.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

ElementID

The element Id. Used if ElementName is empty.

ChildIndex

The index of the child.

Node Output

ChildElementName

The name of the child element.

ChildElementId

The element Id of the child element.

UIe:Element:GetChildByName

Gets the element Id of a child by its name.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

ElementID

The element Id. Used if ElementName is empty.

ChildIndexName

The name of the child element.

Node Output

ChildElementId

The element Id of the child element.

UIe:Element:GetIndexOfChild Node

Gets the index of the specified child.
Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

ElementID

The element Id. Used if ElementName is empty.

ChildElementName

The name of the child element.

ChildElementId

The child element Id. Used if ChildElementName is empty.

Node Output

IndexOfChild

The index of the child element.

Ule:Element:GetNumChildElements Node

Gets the number of children of an element.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

ElementID

The element Id. Used if ElementName is empty.

Node Output

NumChildElements

The number of child elements.

Ule:Element:GetParent Node

Gets the parent of an element.

Node Inputs

Activate

Updates the outputs.
ElementName

Name of the element.

ElementID

The element Id. Used if ElementName is empty.

Node Output

ParentElementName

The name of the parent element.

ParentElementId

The element Id of the parent element.

UI Fader Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard’s new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

Use the following flow graph nodes to perform actions on the fader component.

UIe:Fader:Animation Node

Animates the fader component on the specified element.

Node Inputs

Activate

Starts a fade animation.

ElementName

Name of the fader element.

ElementId

The element Id of the fader element. Used if ElementName is empty.

StartValue

Value at which the fade starts.

Valid values: 0 = Invisible | 1 = Opaque | -1 = Start from the current value

TargetValue

Value at which the fade ends.

Valid values: 0 = Invisible | 1 = Opaque
**Speed**

Rate at which the element fades.

Valid values: 0 = Instant fade | 0.5 = Slow fade | 1 = One second fade | 2 = Fade twice as fast

**Node Outputs**

**OnComplete**

Sends a signal when the fade is complete.

**OnInterrupted**

Sends a signal when the fade is interrupted by another fade starting.

**UIe:Fader:GetFadeValue Node**

Gets the floating-point fade value of an element.

**Node Inputs**

**Activate**

Updates the output.

**ElementName**

Name of the fader element.

**ElementId**

The element Id of the fader element. Used if ElementName is empty.

**Node Output**

**Value**

The floating-point fade value of the element (ElementId).

**UIe:Fader:SetFadeValue Node**

Sets the fade value of an element.

**Node Inputs**

**Activate**

When triggered, assigns Value as the fade value of the fader component of the element.

**ElementName**

Name of the fader element.

**ElementId**

The element Id of the fader element. Used if ElementName is empty.

**Value**

The fade value to assign to the fader component for the element.
**Node Output**

**Done**

Sends a signal when the node's action is finished.

**UIe Image Component Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about **Script Canvas**, Lumberyard's new visual scripting environment.

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 about the latest features, see the **Amazon Lumberyard User Guide**.

Use the following flow graph nodes to perform actions on the image component.

**UIe:Image:GetImageSource Node**

Replaced by **UIe:Image:GetSprite Node (p. 502)**.

Retrieves the texture file path currently used by the specified image element.

**Node Inputs**

Activate

Updates the output.

**ElementName**

Name of the image element.

**Node Outputs**

**Value**

Outputs the file path of the image that is currently on the element.

**UIe:Image:SetImageSource Node**

Replaced by **UIe:Image:SetSprite Node (p. 502)**.

Changes the texture on the specified image element.

**Node Inputs**

Activate

Set the texture.

**ElementName**

Name of the image element.
ImagePath
File path of the texture to display.

Node Output

Done
Sends a signal when the node's action is finished.

UIe:Image:GetSprite Node
Gets the texture file path currently used by the specified image element.

Node Inputs

Activate
Updates the output.

ElementName
Name of the image element.

Node Output

Value
Outputs the file path of the image that is currently on the element.

UIe:Image:SetSprite Node
Sets the texture on the specified image element.

Node Inputs

Activate
Sets the texture.

ElementName
Name of the image element.

ImagePath
File path of the texture to display.

Node Output

Done
Sends a signal when the node's action is finished.

UIe:Image:GetImageType Node
Gets the type of the image. Affects how the texture or sprite is mapped to the image rectangle.
Node Inputs

Activate

Updates the output.

ElementName

Name of the image element.

Node Output

ImageType

An integer representing how the image is scaled and placed.

Valid values: 0 = Stretched | 1 = Sliced | 2 = Fixed | 3 = Tiled | 4 = Stretched to fit | 5 = Stretched to fill

UIe:Image:SetImageType Node

Sets the type of the image. Affects how the texture or sprite is mapped to the image rectangle.

Node Inputs

Activate

Updates the output.

ElementName

Name of the image element.

ImageType

An integer representing how the image is scaled and placed.

Valid values: 0 = Stretched | 1 = Sliced | 2 = Fixed | 3 = Tiled | 4 = Stretched to fit | 5 = Stretched to fill

Node Output

Done

Sends a signal when the node's action is finished.

UIe:Image:GetColor Node

Gets the color tint for the image.

Node Inputs

Activate

Updates the output.

ElementName

Name of the image element.
Node Outputs

Color

The RGB value (0 – 255 each for R, G, and B) of the element (ElementID).

Alpha

The alpha value (0 – 255) of the element (ElementID).

Ule:Image:SetColor Node

Sets the color tint for the image.

Node Inputs

Activate

Updates the output.

ElementName

Name of the image element.

Color

The RGB value (0 – 255 each for R, G, and B).

Alpha

A floating-point alpha value (0 – 255).

Node Output

Done

Sends a signal when the node’s action is finished.

Ule Interactable Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard’s new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

Use the following flow graph node for the Interactable component.

Ule:Interactable:SetIsHandlingEvents Node

Sets the Boolean “is handling events” state of the element.

The Interactable flow graph nodes can be used to get or set values on any interactive UI element.
Interactive UI elements are elements that players can interacted with in game, such as button, text input, check box, slider, and so on. The **SetIsHandlingEvents** flow graph node sets whether an interactive UI element should handle input events. If set to false, then the UI element does not respond to input events, and its visual state is also changed to disabled.

**Node Inputs**

**Activate**
- Sets the "is handling events" state.

**ElementName**
- Name of the element.

**State**
- The Boolean "is handling events" state of the element.

**Node Output**

**Done**
- Sends a signal when the node's action is finished.

**Ule Layout Column Component Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about **Script Canvas**, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

Use the following flow graph nodes to perform actions on the layout column component.

**Ule:LayoutColumn:GetOrder Node**

Gets the vertical order of the **LayoutColumn** component for an element.

**Node Inputs**

**Activate**
- Updates the output.

**ElementName**
- Name of the element.

**Node Output**

**Order**
- An integer representing the vertical order.
Valid values: 0 = Top to bottom | 1 = Bottom to top

**UIe:LayoutColumn:SetOrder Node**

Sets the vertical order of the LayoutColumn component for an element.

**Node Inputs**

*Activate*

Sets the vertical order for the element.

*ElementName*

Name of the element.

*Order*

An integer representing the vertical order. 0 = Top to bottom | 1 = Bottom to top.

**Node Output**

*Done*

Sends a signal when the node's action is finished.

**UIe:LayoutColumn:GetPadding Node**

Gets the padding (in pixels) inside the edges of the LayoutColumn component for an element.

**Node Inputs**

*Activate*

Updates the outputs.

*ElementName*

Name of the element.

**Node Outputs**

*Left*

An integer representing the padding inside the left edge of the element.

*Right*

An integer representing the padding inside the right edge of the element.

*Top*

An integer representing the padding inside the top edge of the element.

*Bottom*

An integer representing the padding inside the bottom edge of the element.

**UIe:LayoutColumn:SetPadding Node**

Sets the padding (in pixels) inside the edges of the LayoutColumn component for an element.
Node Inputs

**Activate**
Updates the outputs.

**ElementName**
Name of the element.

**Left**
An integer representing the padding inside the left edge of the element.

**Right**
An integer representing the padding inside the right edge of the element.

**Top**
An integer representing the padding inside the top edge of the element.

**Bottom**
An integer representing the padding inside the bottom edge of the element.

Node Output

**Done**
Sends a signal when the node’s action is finished.

**UIe:LayoutColumn:GetSpacing Node**
Gets the spacing (in pixels) between child elements of the `LayoutColumn` component for an element.

Node Inputs

**Activate**
Updates the output.

**ElementName**
Name of the element.

Node Output

**Spacing**
A float value of the spacing (in pixels) between child elements of the element (`ElementName`).

**UIe:LayoutColumn:SetSpacing Node**
Sets the spacing (in pixels) between child elements of the `LayoutColumn` component for an element.

Node Inputs

**Activate**
Updates the output.

**ElementName**
Name of the element.
Spacing

A float value of the spacing (in pixels) between child elements of the element (ElementName).

Node Output

Done

Sends a signal when the node's action is finished.

UI Layout Grid Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

Use the following flow graph nodes to perform actions on the layout grid component.

**UI:LayoutGrid:GetCellSize Node**

Gets the size (in pixels) of a child element in the layout.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Outputs

CellWidth

The width (in pixels) of a child element of element (ElementID).

CellHeight

The height (in pixels) of a child element of element (ElementID).

**UI:LayoutGrid:SetCellSize Node**

Sets the size (in pixels) of a child element in the layout.

Node Inputs

Activate

Updates the output.
**ElementName**

Name of the element.

**CellWidth**

The width (in pixels) of a child element of element (ElementID).

**CellHeight**

The height (in pixels) of a child element of element (ElementID).

**Node Output**

**Done**

Sends a signal when the node's action is finished.

**UIe:LayoutGrid:GetHorizontalOrder Node**

Gets the horizontal order for the layout.

**Node Inputs**

**Activate**

Updates the outputs.

**ElementName**

Name of the element.

**Node Output**

**Order**

An integer representing the horizontal order.

Valid values: 0 = Left to right | 1 = Right to left

**UIe:LayoutGrid:SetHorizontalOrder Node**

Sets the horizontal order for the layout.

**Node Inputs**

**Activate**

Updates the outputs.

**ElementName**

Name of the element.

**Order**

An integer representing the horizontal order.

Valid values: 0 = Left to right | 1 = Right to left
Node Output

Done
Sends a signal when the node's action is finished.

Ule:LayoutGrid:GetPadding Node
Gets the padding (in pixels) inside the edges of the LayoutGrid component for an element.

Node Inputs

Activate
Updates the output.

ElementName
Name of the element.

Node Output

Left
An integer representing the padding inside the left edge of the element.

Right
An integer representing the padding inside the right edge of the element.

Top
An integer representing the padding inside the top edge of the element.

Bottom
An integer representing the padding inside the bottom edge of the element.

Ule:LayoutGrid:SetPadding Node
Sets the padding (in pixels) inside the edges of the LayoutGrid component for an element.

Node Inputs

Activate
Updates the output.

ElementName
Name of the element.

Left
An integer representing the padding inside the left edge of the element.

Right
An integer representing the padding inside the right edge of the element.

Top
An integer representing the padding inside the top edge of the element.

Bottom
An integer representing the padding inside the bottom edge of the element.
Node Output

Done

Sends a signal when the node's action is finished.

_Ule:LayoutGrid:GetSpacing Node_

Gets the spacing (in pixels) between child elements of the LayoutGrid component for an element.

Node Inputs

_Activate_

Updates the output.

_ElementName_

Name of the element.

Node Output

Spacing

A float value of the spacing (in pixels) between child elements of the element (ElementID).

_Ule:LayoutGrid:SetSpacing Node_

Sets the spacing (in pixels) between child elements of the LayoutGrid component for an element.

Node Inputs

_Activate_

Updates the output.

_ElementName_

Name of the element.

Spacing

A float value of the spacing (in pixels) between child elements of the element (ElementID).

Node Output

Done

Sends a signal when the node's action is finished.

_Ule:LayoutGrid:GetStartingDirection Node_

Gets the starting direction for the layout.

Node Inputs

_Activate_

Updates the output.
**ElementName**

Name of the element.

**Node Output**

**Direction**

An integer representing the direction.

Valid values: 0 = Horizontal order | 1 = Vertical order

**Ule:LayoutGrid:SetStartingDirection Node**

Sets the starting direction for the layout.

**Node Inputs**

**Activate**

Set the starting direction for the layout.

**ElementName**

Name of the element.

**Direction**

An integer representing the horizontal order.

Valid values: 0 = Horizontal order | 1 = Vertical order.

**Node Output**

**Done**

Sends a signal when the node's action is finished.

**Ule:LayoutGrid:GetVerticalOrder Node**

Gets the vertical order for the layout.

**Node Inputs**

**Activate**

Updates the outputs.

**ElementName**

Name of the element.

**Node Output**

**Action**

An integer representing the vertical order.

Valid values: 0 = Top to bottom | 1 = Bottom to top
**UI Flow Graph Nodes**

**UIe:LayoutGrid:SetVerticalOrder Node**
Sets the vertical order for the layout.

**Node Inputs**

**Activate**
- Sets the vertical order for the layout.

**ElementName**
- Name of the element.

**Action**
- An integer representing the vertical order.
  - Valid values: 0 = Top to bottom | 1 = Bottom to top

**Node Output**

**Done**
- Sends a signal when the node's action is finished.

**UIe Layout Row Component Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about [Script Canvas](https://docs.amazon.lumberyard.com/ScriptCanvas/), Lumberyard's new visual scripting environment.

This topic references tools and features that are **legacy**. If you want to use legacy tools in Lumberyard Editor, disable the [CryEntity Removal gem](https://docs.amazon.lumberyard.com/EntityRemoval) using the [Project Configurator](https://docs.amazon.lumberyard.com/ProjectConfigurator/) or the command line. To learn about the latest features, see the [Amazon Lumberyard User Guide](https://docs.amazon.lumberyard.com/). Use the following flow graph nodes to perform actions on the layout row component.

**UIe:LayoutRow:GetOrder Node**
Gets the horizontal order of the **LayoutRow** component for an element.

**Node Inputs**

**Activate**
- Updates the output.

**ElementName**
- Name of the element.

**Node Output**

**Order**
- An integer representing the horizontal order.
  - Valid values: 0 = Left to right | 1 = Right to left
**UIe:LayoutRow:SetOrder Node**

Sets the horizontal order of the **LayoutRow** component for an element.

**Node Inputs**

- **Activate**
  Updates the output.
- **ElementName**
  Name of the element.
- **Order**
  An integer representing the horizontal order.
  Valid values: \(0 = \text{Left to right} \mid 1 = \text{Right to left}\)

**Node Output**

- **Done**
  Sends a signal when the node's action is finished.

**UIe:LayoutRow:GetPadding Node**

Gets the padding (in pixels) inside the edges of the **LayoutRow** component for an element.

**Node Inputs**

- **Activate**
  Updates the output.
- **ElementName**
  Name of the element.

**Node Outputs**

- **Left**
  An integer representing the padding inside the left edge of the element.
- **Right**
  An integer representing the padding inside the right edge of the element.
- **Top**
  An integer representing the padding inside the top edge of the element.
- **Bottom**
  An integer representing the padding inside the bottom edge of the element.

**UIe:LayoutRow:SetPadding Node**

Sets the padding (in pixels) inside the edges of the **LayoutRow** component for an element.
Node Inputs

Activate

Sets the padding (in pixels) inside the edges of the LayoutRow.

ElementName

Name of the element.

Left

An integer representing the padding inside the left edge of the element.

Right

An integer representing the padding inside the right edge of the element.

Top

An integer representing the padding inside the top edge of the element.

Bottom

An integer representing the padding inside the bottom edge of the element.

Node Output

Done

Sends a signal when the node's action is finished.

UIe:LayoutRow:GetSpacing Node

Gets the spacing (in pixels) between child elements of the LayoutRow component for an element.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Node Output

Spacing

A float value of the spacing (in pixels) between child elements of the element (ElementName).

UIe:LayoutRow:SetSpacing Node

Sets the spacing (in pixels) between child elements of the LayoutRow component for an element.

Node Inputs

Activate

Sets the spacing (in pixels) between child elements.

ElementName

Name of the element.
Spacing

A float value of the spacing (in pixels) between child elements of the element \( \text{ElementName} \).

Node Output

Done
Sends a signal when the node's action is finished.

UIe Mask Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

Use the following flow graph nodes to perform actions on the mask component.

**UIe:Mask:GetDrawBehind Node**

Gets whether mask is drawn behind the child elements.

Node Inputs

Activate
Updates the outputs.

**ElementName**
Name of the element.

Node Output

**DrawBehind**
Indicates whether mask is drawn behind the child elements.

**UIe:Mask:SetDrawBehind Node**

Sets whether mask is drawn behind the child elements.

Node Inputs

Activate
Sets whether mask is drawn behind the child elements.

**ElementName**
Name of the element.

**DrawBehind**
Sets whether mask is drawn behind the child elements.
Node Output

Done
Sends a signal when the node's action is finished.

UIe:Mask:GetDrawInFront Node
Gets whether mask is drawn in front of child elements.

Node Inputs

Activate
Updates the outputs.

ElementName
Name of the element.

Node Output

DrawInFront
Indicates whether mask is drawn in front of child elements.

UIe:Mask:SetDrawInFront Node
Sets whether mask is drawn in front of child elements.

Node Inputs

Activate
Sets whether mask is drawn in front of child elements.

ElementName
Name of the element.

DrawInFront
Sets whether mask is drawn in front of child elements.

Node Output

Done
Sends a signal when the node's action is finished.

UIe:Mask:GetIsMaskingEnabled Node
Gets whether masking is enabled.

Node Inputs

Activate
Updates the outputs.
ElementName

Name of the element.

Node Output

IsMaskingEnabled

Indicates whether masking is enabled.

UIe:Mask:SetIsMaskingEnabled Node

Sets whether masking is enabled.

Node Inputs

Activate

Sets whether masking is enabled.

ElementName

Name of the element.

IsMaskingEnabled

Sets whether masking is enabled.

Node Output

Done

Sends a signal when the node's action is finished.

UIe:Mask:GetUseAlphaTest Node

Gets whether to use the alpha channel in the mask visual's texture to define the mask.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Node Output

UseAlphaTest

Indicates whether to use the alpha channel in the mask visual's texture to define the mask.

UIe:Mask:SetUseAlphaTest Node

Sets whether to use the alpha channel in the mask visual's texture to define the mask.
Node Inputs

Activate

Sets whether to use the alpha channel in the mask visual’s texture to define the mask.

ElementName

Name of the element.

UseAlphaTest

Sets whether to use the alpha channel in the mask visual’s texture to define the mask.

Node Output

Done

Sends a signal when the node’s action is finished.

UIe ScrollBox Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard’s new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

Use the following flow graph nodes to perform actions on the ScrollBox component.

UIe:ScrollBox:FindClosestContentChildElement Node

Finds the child of the content element that is closest to the content anchors.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Node Output

ClosestElement

The element currently closest to the focused element.

UIe:ScrollBox:GetContentEntity Node

Gets the content element for the ScrollBox.
Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Node Output

Content

The element that the ScrollBox scrolls.

Ule:ScrollBox:SetContentEntity Node

Sets the content element for the ScrollBox.

Node Inputs

Activate

Sets the content element for the ScrollBox.

ElementName

Name of the element.

Content

The element that the ScrollBox scrolls.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:ScrollBox:GetIsHorizontalScrollingEnabled Node

Gets whether the ScrollBox allows horizontal scrolling.

Node Inputs

Activate

Updates the outputs.

ElementName

Name of the element.

Node Output

Enabled

Indicates whether horizontal scrolling is enabled.
**Ule:ScrollBox:SetIsHorizontalScrollingEnabled Node**

Sets whether the **ScrollBox** allows horizontal scrolling.

**Node Inputs**

*Activate*

- Sets whether the **ScrollBox** allows horizontal scrolling.

*ElementName*

- Name of the element.

*Enabled*

- Sets whether horizontal scrolling is enabled.

**Node Output**

*Done*

- Sends a signal when the node's action is finished.

**Ule:ScrollBox:GetIsScrollingConstrained Node**

Gets whether the **ScrollBox** restricts scrolling to the content area.

**Node Inputs**

*Activate*

- Updates the outputs.

*ElementName*

- Name of the element.

**Node Output**

*IsConstrained*

- Indicates whether scrolling is constrained.

**Ule:ScrollBox:SetIsScrollingConstrained Node**

Sets whether the **ScrollBox** restricts scrolling to the content area.

**Node Inputs**

*Activate*

- Sets whether the **ScrollBox** restricts scrolling to the content area.

*ElementName*

- Name of the element.

*IsConstrained*

- Sets whether scrolling is constrained.
Node Output

Done

Sends a signal when the node's action is finished.

**UIe:ScrollBox:GetIsVerticalScrollingEnabled Node**

Gets whether the **ScrollView** allows vertical scrolling.

Node Inputs

**Activate**

Updates the output.

**ElementName**

Name of the element.

Node Output

**Enabled**

Indicates whether vertical scrolling is enabled.

**UIe:ScrollBox:SetIsVerticalScrollingEnabled Node**

Sets whether the **ScrollView** allows vertical scrolling.

Node Inputs

**Activate**

Sets whether the **ScrollView** allows vertical scrolling.

**ElementName**

Name of the element.

**Enabled**

Sets whether vertical scrolling is enabled.

Node Output

**Done**

Sends a signal when the node's action is finished.

**UIe:ScrollBox:GetScrollOffset Node**

Gets the scroll offset of the **ScrollView**.

Node Inputs

**Activate**

Updates the outputs.
ElementName
Name of the element.

Node Output
HorizOffset
The horizontal scroll offset of the element identified by ElementName.
VertOffset
The vertical scroll offset of the element identified by ElementName.

UIe:ScrollBox:SetScrollOffset Node
Sets the scroll offset of the ScrollBox.

Node Inputs
Activate
Sets the scroll offset of the ScrollBox.
ElementName
Name of the element.
HorizOffset
The horizontal scroll offset of ElementName.
VertOffset
The vertical scroll offset of ElementName.

Node Output
Done
Sends a signal when the node’s action is finished.

UIe:ScrollBox:GetScrollOffsetChangedActionName Node
Gets the action triggered when the ScrollBox drag is completed.

Node Inputs
Activate
Updates the output.
ElementName
Name of the element.

Node Output
ChangedAction
The action name.
Ule:ScrollBox:SetScrollOffsetChangedActionName Node
Sets the action triggered when the ScrollBox drag is completed.

Node Inputs
Activate
Sets the action triggered when the ScrollBox drag is completed.
ElementName
Name of the element.
ChangedAction
The action name.

Node Output
Done
Sends a signal when the node's action is finished.

Ule:ScrollBox:GetScrollOffsetChangingActionName Node
Gets the action triggered while the ScrollBox is being dragged.

Node Inputs
Activate
Updates the output.
ElementName
Name of the element.

Node Output
ChangingAction
The action name.

Ule:ScrollBox:SetScrollOffsetChangingActionName Node
Sets the action triggered while the ScrollBox is being dragged.

Node Inputs
Activate
Sets the action triggered while the ScrollBox is being dragged.
ElementName
Name of the element.
ChangingAction
The action name.
Node Output

Done

Sends a signal when the node's action is finished.

**UIe:ScrollBox:GetSnapGrid Node**

Get the snapping grid of the **ScrollBox**.

Node Inputs

Activate

Updates the outputs.

**ElementName**

Name of the element.

Node Outputs

**HorizSpacing**

The horizontal grid spacing of the element identified by **ElementName**.

**VertSpacing**

The vertical grid spacing of the element identified by **ElementName**.

**UIe:ScrollBox:SetSnapGrid Node**

Sets the snapping grid of the **ScrollBox**.

Node Inputs

Activate

Sets the snapping grid of the **ScrollBox**.

**ElementName**

Name of the element.

**HorizSpacing**

The horizontal grid spacing of the element identified by **ElementName**.

**VertSpacing**

The vertical grid spacing of the element identified by **ElementName**.

Node Output

**Done**

Sends a signal when the node's action is finished.

**UIe:ScrollBox:GetSnapMode Node**

Gets the snap mode for the **ScrollBox**.
Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

SnapMode

An integer representing the snap mode state.

Valid values:
0 = None | 1 = Children | 2 = Grid

UIe:ScrollBox:SetSnapMode Node

Sets the snap mode for the ScrollBox.

Node Inputs

Activate

Sets the snap mode for the ScrollBox.

ElementName

Name of the element.

SnapMode

An integer representing the snap mode state.

Valid values:
0 = None | 1 = Children | 2 = Grid

Node Output

Done

Sends a signal when the node's action is finished.

UIe:ScrollBox:GetHorizontalScrollBarVisibility Node

Gets horizontal scroll bar visibility behavior.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.
**Node Output**

**ScrollBarVisibility**
An integer that represents the scroll bar visibility behavior.

Valid values: 0 = AlwaysVisible | 1 = AutoHide | 2 = AutoHideAndResizeViewArea

**Ule:ScrollBox:SetHorizontalScrollBarVisibility Node**
Sets horizontal scroll bar visibility behavior.

**Node Inputs**

**Activate**
Sets horizontal scroll bar visibility behavior.

**CanvasID**
Unique identifier of the element's canvas.

**ElementID**
Unique identifier of the element.

**ScrollBarVisibility**
An integer representing the scroll bar visibility behavior.

0 = AlwaysVisible | 1 = AutoHide | 2 = AutoHideAndResizeViewArea

**Node Output**

**Done**
Sends a signal when the node's action is finished.

**Ule:ScrollBox:GetVerticalScrollBarVisibility Node**
Gets vertical scroll bar visibility behavior.

**Node Inputs**

**Activate**
Updates the outputs.

**CanvasID**
Unique identifier of the element's canvas.

**ElementID**
Unique identifier of the element.

**Node Output**

**ScrollBarVisibility**
An integer that represents the scroll bar visibility behavior.
Valid values: 0 = AlwaysVisible | 1 = AutoHide | 2 = AutoHideAndResizeViewArea

Ule:ScrollBox:SetVerticalScrollBarVisibility Node
Sets vertical scroll bar visibility behavior.

Node Inputs
Activate
Sets vertical scroll bar visibility behavior.
CanvasID
Unique identifier of the element's canvas.
ElementID
Unique identifier of the element.
ScrollBarVisibility
An integer representing the scroll bar visibility behavior.
0 = AlwaysVisible | 1 = AutoHide | 2 = AutoHideAndResizeViewArea.

Node Output
Done
Sends a signal when the node's action is finished.

Ule:ScrollBox:GetHorizontalScrollBarEntity Node
Gets the horizontal scroll bar element for the ScrollBox.

Node Inputs
Activate
Updates the outputs.
CanvasID
Unique identifier of the element's canvas.
ElementID
Unique identifier of the element.

Node Output
HorizontalScrollBar
The element that scrolls the ScrollBox horizontally.

Ule:ScrollBox:SetHorizontalScrollBarEntity Node
Sets the horizontal scroll bar element for the ScrollBox.
**Node Inputs**

**Activate**
- Sets the horizontal scroll bar element for the **ScrollBox**.

**CanvasID**
- Unique identifier of the element's canvas.

**ElementID**
- Unique identifier of the element.

**HorizontalScrollBar**
- The element that scrolls the **ScrollBox** horizontally.

**Node Output**

**Done**
- Sends a signal when the node's action is finished.

**UIe:ScrollBox:GetVerticalScrollBarEntity Node**
Gets the vertical scroll bar element for the **ScrollBox**.

**Node Inputs**

**Activate**
- Updates the outputs.

**CanvasID**
- Unique identifier of the element's canvas.

**ElementID**
- Unique identifier of the element.

**Node Output**

**VerticalScrollBar**
- The element that scrolls the **ScrollBox** vertically.

**UIe:ScrollBox:SetVerticalScrollBarEntity Node**
Sets the vertical scroll bar element for the **ScrollBox**.

**Node Inputs**

**Activate**
- Sets the vertical scroll bar element for the **ScrollBox**.

**CanvasID**
- Unique identifier of the element's canvas.

**ElementID**
- Unique identifier of the element.
VerticalScrollBar

The element that scrolls the ScrollBox vertically.

Node Output

Done

Sends a signal when the node's action is finished.

Ule ScrollBar Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

Use the following flow graph nodes to perform actions on the Scrollbar component.

Ule:Scrollbar:GetHandleEntity Node

Gets the handle element of the scroll bar.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementName

Name of the element.

Node Output

Handle

The handle element.

Ule:Scrollbar:SetHandleEntity Node

Sets the handle element of the scroll bar.

Node Inputs

Activate

Sets the handle element.
CanvasID
Unique identifier of the element's canvas.

ElementName
Name of the element.

Handle
The handle element.

Node Output

Done
Sends a signal when the node's action is finished.

Ule:Scrollbar:GetValue Node
Gets the value of the scrollbar.

Node Inputs

Activate
Updates the output.

CanvasID
Unique identifier of the element's canvas.

ElementName
Name of the element.

Node Output

Value
The scrollbar value of the element identified by ElementName.

Ule:Scrollbar:SetValue Node
Sets the value of the scrollbar.

Node Inputs

Activate
Sets the value of the scrollbar.

CanvasID
Unique identifier of the element's canvas.

ElementName
Name of the element.

Value
The scrollbar value of the element identified by ElementName.
Node Output

Done

Sends a signal when the node's action is finished.

Ule:Scrollbar:GetHandleSize Node

Gets the size of the handle.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementName

Name of the element.

Node Output

HandleSize

The size of the handle of the element identified by ElementName.

Ule:Scrollbar:SetHandleSize Node

Sets the size of the handle.

Node Inputs

Activate

Sets the size of the handle.

CanvasID

Unique identifier of the element's canvas.

ElementName

Name of the element.

HandleSize

The size of the handle of the element identified by ElementName.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Scrollbar:GetMinHandlePixelSize Node

Gets the minimum size in pixels of the handle.
Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementName

Name of the element.

Node Output

MinHandleSize

The minimum size in pixels of the handle of the element identified by ElementName.

UIe:Scrollbar:SetMinHandlePixelSize Node

Sets the minimum size in pixels of the handle.

Node Inputs

Activate

Sets the minimum size in pixels of the handle.

CanvasID

Unique identifier of the element's canvas.

ElementName

Name of the element.

MinHandleSize

The minimum size in pixels of the handle of the element identified by ElementName.

Node Output

Done

Sends a signal when the node's action is finished.

UIe:Scrollbar:GetValueChangedActionName Node

Gets the action triggered when the value is done changing.
**ElementName**

Name of the element.

**Node Output**

**ValueChangedAction**

The action name.

**Ule:Scrollbar:SetValueChangedActionName Node**

Sets the action triggered when the value is done changing.

**Node Inputs**

**Activate**

Updates the output.

**CanvasID**

Unique identifier of the element's canvas.

**ElementName**

Name of the element.

**ValueChangedAction**

The action name.

**Node Output**

**Done**

Sends a signal when the node's action is finished.

**Ule:Scrollbar:GetValueChangingActionName Node**

Gets the action triggered while the value is changing.

**Node Inputs**

**Activate**

Updates the output.

**CanvasID**

Unique identifier of the element's canvas.

**ElementName**

Name of the element.

**ValueChangedAction**

The action name.
**UI Flow Graph Nodes**

**UIe:Scrollbar:SetValueChangingActionName Node**
Sets the action triggered while the value is changing.

**Node Inputs**

- **Activate**
  Updates the output.
- **CanvasID**
  Unique identifier of the element's canvas.
- **ElementName**
  Name of the element.
- **ValueChangingAction**
  The action name.

**Node Output**

- **Done**
  Sends a signal when the node's action is finished.

**UIe Slider Component Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

Use the following flow graph nodes to perform actions on the slider component.

**UIe:Slider:GetFillEntity Node**
Gets the fill element.

**Node Inputs**

- **Activate**
  Updates the output.
- **ElementName**
  Name of the element.

**Node Output**

- **FillElement**
  The fill element.
**UIe:Slider:SetFillEntity Node**

Sets the fill element.

**Node Inputs**

- **Activate**
  - Sets the fill element.
- **ElementName**
  - Name of the element.
- **FillElement**
  - The fill element.

**Node Output**

- **Done**
  - Sends a signal when the node's action is finished.

**UIe:Slider:GetManipulatorEntity Node**

Gets the manipulator element.

**Node Inputs**

- **Activate**
  - Updates the output.
- **ElementName**
  - Name of the element.

**Node Output**

- **ManipulatorElement**
  - The manipulator element.

**UIe:Slider:SetManipulatorEntity Node**

Sets the manipulator element.

**Node Inputs**

- **Activate**
  - Sets the manipulator element.
- **ElementName**
  - Name of the element.
- **ManipulatorElement**
  - The manipulator element.
Node Output

Done

Sends a signal when the node's action is finished.

Ule:Slider:GetMaxValue Node

Gets the maximum value of the slider.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

MaxValue

The slider maximum value of the element identified by ElementName.

Ule:Slider:SetMaxValue Node

Sets the maximum value of the slider.

Node Inputs

Activate

Sets the maximum value of the slider.

ElementName

Name of the element.

MaxValue

The slider maximum value of the element identified by ElementName.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:Slider:GetMinValue Node

Gets the minimum value of the slider.

Node Inputs

Activate

Updates the output.
**ElementName**

Name of the element.

**Node Output**

**MinValue**

The slider minimum value of the element identified by **ElementName**.

**UIe:Slider:SetMinValue Node**

Sets the minimum value of the slider.

**Node Inputs**

**Activate**

Sets the minimum value of the slider.

**ElementName**

Name of the element.

**MinValue**

The slider minimum value of the element identified by **ElementName**.

**Node Output**

**Done**

Sends a signal when the node's action is finished.

**UIe:Slider:GetStepValue Node**

Gets the smallest increment allowed between values. Zero means no restriction.

**Node Inputs**

**Activate**

Updates the output.

**ElementName**

Name of the element.

**Node Output**

**StepValue**

The smallest increment allowed between values of the element identified by **ElementName**. Zero means no restriction.

**UIe:Slider:SetStepValue Node**

Sets the smallest increment allowed between values. Zero means no restriction.
**Node Inputs**

**Activate**

Sets the smallest increment allowed between values. Zero means no restriction.

**ElementName**

Name of the element.

**StepValue**

The smallest increment allowed between values of the element identified by `ElementName`. Zero means no restriction.

**Node Output**

**Done**

Sends a signal when the node's action is finished.

**Ule:Slider:GetTrackEntity Node**

Gets the track element.

**Node Inputs**

**Activate**

Updates the output.

**ElementName**

Name of the element.

**Node Output**

**Track**

The track element.

**Ule:Slider:SetTrackEntity Node**

Sets the track element.

**Node Inputs**

**Activate**

Sets the track element.

**ElementName**

Name of the element.

**Track**

The track element.
**Node Output**

**Done**

Sends a signal when the node's action is finished.

**Ule:Slider:GetValue Node**

Gets the value of slider.

**Node Inputs**

**Activate**

Updates the output.

**ElementName**

Name of the element.

**Node Output**

**Value**

The slider value of the element identified by *ElementName*.

**Ule:Slider:SetValue Node**

Sets the value of the slider.

**Node Inputs**

**Activate**

Sets the value of the slider.

**ElementName**

Name of the element.

**Value**

The slider value of the element identified by *ElementName*.

**Node Output**

**Done**

Sends a signal when the node's action is finished.

**Ule:Slider:GetValueChangedActionName Node**

 Gets the action triggered when the value is done changing.

**Node Inputs**

**Activate**

Updates the output.
**ElementName**

Name of the element.

**Node Output**

**ValueChangedAction**

The action name.

**Ule:Slider:SetValueChangedActionName Node**

Sets the action triggered when the value is done changing.

**Node Inputs**

**Activate**

Updates the output.

**ElementName**

Name of the element.

**ValueChangedAction**

The action name.

**Node Output**

**Done**

Sends a signal when the node's action is finished.

**Ule:Slider:GetValueChangingActionName Node**

Gets the action triggered while the value is changing.

**Node Inputs**

**Activate**

Updates the output.

**ElementName**

Name of the element.

**ValueChangedAction**

The action name.

**Node Output**

**ValueChangingAction**

The action name.

**Ule:Slider:SetValueChangingActionName Node**

Sets the action triggered while the value is changing.
Node Inputs

**Activate**

Updates the output.

**ElementName**

Name of the element.

**ValueChangingAction**

The action name.

Node Output

**Done**

Sends a signal when the node's action is finished.

UIe Text Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

Use the following flow graph nodes to perform actions on the text component.

**UIe:Text:GetColor Node**

Gets the color to draw the text string.

Node Inputs

**Activate**

Updates the outputs.

**ElementName**

Name of the element.

Node Outputs

**Color**

The RGB value (0 – 255 each for R, G, and B) of the element identified by **ElementName**.

**Alpha**

The alpha value (0 – 255) of the element identified by **ElementName**.

**UIe:Text:SetColor Node**

Sets the color to draw the text string.
Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Color

The RGB value (0 – 255 each for R, G, and B) of the element identified by ElementName.

Alpha

The alpha value (0 – 255) of the element identified by ElementName.

Node Output

Done

Sends a signal when the node's action is finished.

UIe:Text:GetFont Node

Gets the path to the font.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

Font

The path to the font used by the element.

UIe:Text:SetFontNode

Sets the path to the font.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Font

The path to the font used by the element identified by ElementName.
Node Output

Done

Sends a signal when the node's action is finished.

_Ule:Text:GetFontSize Node_

Gets the font size in points.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

FontSize

The font size of the element identified by `ElementName`.

_Ule:Text:SetFontSize Node_

Sets the font size in points.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

FontSize

The font size of the element identified by `ElementName`.

Node Output

Done

Sends a signal when the node's action is finished.

_Ule:Text:GetOverflowMode Node_

Gets the overflow behavior of the text.

Node Inputs

Activate

Updates the output.
ElementName

Name of the element.

Node Output

OverflowMode

An integer representing how overflow text is handled.

Valid values: 0 = Overflow text | 1 = Clip text

Ule:Text:SetOverflowModeNode

Sets the overflow behavior of the text.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

OverflowMode

An integer representing how overflow text is handled.

Valid values: 0 = Overflow text | 1 = Clip text

Node Output

Done

Sends a signal when the node’s action is finished.

Ule:Text:GetText Node

Gets the text string that the element displays.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

Value

The text string being displayed by the element identified by ElementName.
**UIe:Text:SetText Node**

Sets the text string being displayed by the element.

**Node Inputs**

**Activate**

Updates the output.

**ElementName**

Name of the element.

**Value**

The text string being displayed by the element identified by `ElementName`.

**Node Output**

**Done**

Sends a signal when the node's action is finished.

**UIe:Text:GetWrapText Node**

Gets whether text is wrapped.

**Node Inputs**

**Activate**

Updates the output.

**ElementName**

Name of the element.

**Node Output**

**WrapTextSetting**

An integer representing how long text lines are handled.

Valid values: 0 = No wrap | 1 = Wrap

**UIe:Text:SetWrapText Node**

Gets whether text is wrapped.

**Node Inputs**

**Activate**

Updates the outputs.

**ElementName**

Name of the element.
WrapTextSetting

An integer representing how long text lines are handled.

Valid values: 0 = No wrap | 1 = Wrap

Node Output

Done

Sends a signal when the node's action is finished.

UI Text Input Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

Use the following flow graph nodes to perform actions on the text input component.

**UI:TextInput:GetChangeAction Node**

Gets the action triggered when the text is changed.

**Node Inputs**

Activate

Updates the output.

ElementName

Name of the element.

**Node Output**

ChangeAction

The action name.

**UI:TextInput:SetChangeAction Node**

Sets the action triggered when the text is changed.

**Node Inputs**

Activate

Updates the output.

ElementName

Name of the element.
ChangeAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:TextInput:GetCursorBlinkInterval Node

Gets the cursor blink interval of the text input.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

CursorBlinkInterval

The cursor blink in interval of the element identified by ElementName.

Ule:TextInput:SetCursorBlinkInterval Node

Gets the cursor blink interval of the text input.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

CursorBlinkInterval

The cursor blink in interval of the element identified by ElementName.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:TextInput:GetEndEditAction Node

Gets the action triggered when the editing of text is finished.
Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

EndEditAction

The action name.

UIe:TextInput:SetEndEditAction Node

Sets the action triggered when the editing of text is finished.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

Done

Sends a signal when the node's action is finished.

UIe:TextInput:GetEnterAction Node

Gets the action triggered when Enter is pressed.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

EnterAction

The action name.
UIe:TextInput:SetEnterAction Node

Sets the action triggered when **Enter** is pressed.

**Node Inputs**

**Activate**

Updates the output.

**ElementName**

Name of the element.

**EnterAction**

The action name.

**Node Output**

**Done**

Sends a signal when the node's action is finished.

UIe:TextInput:GetIsPasswordField Node

Gets whether the text input is configured as a password field.

**Node Inputs**

**Activate**

Updates the output.

**ElementName**

Name of the element.

**Node Output**

**IsPasswordField**

Boolean. Whether the element identified by **ElementName** is configured as a password field.

UIe:TextInput:SetIsPasswordField Node

Sets whether the text input is configured as a password field.

**Node Inputs**

**Activate**

Updates the output.

**ElementName**

Name of the element.

**IsPasswordField**

Boolean. Whether the element identified by **ElementName** is configured as a password field.
Node Output

Done

Sends a signal when the node's action is finished.

UIe:TextInput:GetMaxStringLength Node

Gets the maximum number of characters that can be entered.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

MaxStringLength

An integer representing the maximum number of characters that can be entered.

Valid values: 0 = none allowed | -1 = unlimited

UIe:TextInput:SetMaxStringLength Node

Sets the maximum number of characters that can be entered.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

MaxStringLength

An integer representing the maximum number of characters that can be entered.

Valid values: 0 = none allowed | -1 = unlimited

Node Output

Done

Sends a signal when the node's action is finished.

UIe:TextInput:GetPlaceholderTextEntity Node

Gets the placeholder text element.
Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Output

PlaceHolderTextElement

The placeholder text element.

Ule:TextInput:SetPlaceHolderTextEntity Node

Sets the placeholder text element.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

PlaceHolderTextElement

The placeholder text element.

Node Output

Done

Sends a signal when the node's action is finished.

Ule:TextInput:GetText Node

Gets the text string that the element is displaying or allowing to be edited.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Value

The text string being displayed or edited by the element
**Ule:TextInput:SetText Node**

Sets the text string that the element is displaying or allowing to be edited.

**Node Inputs**

**Activate**

Updates the output.

**ElementName**

Name of the element.

**Value**

The text string being displayed or edited by the element

**Node Output**

**Done**

Sends a signal when the node's action is finished.

**Ule:TextInput:GetTextCursorColor Node**

Gets the color to be used for the text cursor.

**Node Inputs**

**Activate**

Updates the output.

**ElementName**

Name of the element.

**Node Outputs**

**Color**

The RGB value (0 – 255 each for R, G, and B) of the element identified by **ElementName**.

**Alpha**

The alpha value (0 – 255) of the element identified by **ElementName**.

**Ule:TextInput:SetTextCursorColor Node**

Sets the color to be used for the text cursor.

**Node Inputs**

**Activate**

Updates the output.

**ElementName**

Name of the element.
**Color**

The RGB value (0 – 255 each for R, G, and B) of the element identified by `ElementName`.

**Alpha**

The alpha value (0 – 255) of the element identified by `ElementName`.

**Node Output**

**Done**

Sends a signal when the node's action is finished.

**UIe:TextInput:GetTextEntity Node**

Gets the text element.

**Node Inputs**

**Activate**

Updates the output.

**ElementName**

Name of the element.

**Node Output**

**TextElement**

The text element.

**UIe:TextInput:SetTextEntity Node**

Gets the text element.

**Node Inputs**

**Activate**

Updates the output.

**ElementName**

Name of the element.

**TextElement**

The text element.

**Node Output**

**Done**

Sends a signal when the node's action is finished.

**UIe:TextInput:GetTextSelectionColor Node**

Gets the color to be used for the text background when it is selected.
Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Node Outputs

Color

The RGB value (0 – 255 each for R, G, and B) of the element identified by ElementName.

Alpha

The alpha value (0 – 255) of the element identified by ElementName.

UIe:TextInput:SetTextSelectionColor Node

Gets the color to be used for the text background when it is selected.

Node Inputs

Activate

Updates the output.

ElementName

Name of the element.

Color

The RGB value (0 – 255 for R, G, and B) of the element identified by ElementName.

Alpha

The alpha value (0 – 255) of the element identified by ElementName.

Node Output

Done

Sends a signal when the node's action is finished.

UI Transform Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.
Use the following flow graph nodes to perform actions on the transform component.

**Ule:Transform:GetCanvasPosition Node**

Gets the position of an element in canvas space.

**Node Inputs**

- **Activate**
  
  Updates the outputs.

- **ElementName**
  
  Name of the element.

- **ElementID**
  
  The element Id. Used if `ElementName` is empty.

**Node Outputs**

- **XPosition**
  
  The x position of the element in canvas space.

- **YPosition**
  
  The y position of the element in canvas space.

**Ule:Transform:SetCanvasPosition Node**

Sets the position of an element in canvas space.

**Node Inputs**

- **Activate**
  
  Sets the position of the element in canvas space.

- **ElementName**
  
  Name of the element.

- **ElementID**
  
  The element Id. Used if `ElementName` is empty.

- **XPosition**
  
  The x position of the element in canvas space.

- **YPosition**
  
  The y position of the element in canvas space.

**Node Outputs**

- **Done**
  
  Sends a signal when the node’s action is finished.
UIe:Transform:GetLocalPosition Node

Gets the relative position of an element from the center of the element's anchors.

**Node Inputs**

**Activate**
- Updates the output.

**ElementName**
- Name of the element.

**ElementID**
- The element Id. Used if `ElementName` is empty.

**Node Outputs**

**XPosition**
- The relative x position of the element.

**YPosition**
- The relative y position of the element.

UIe:Transform:SetLocalPosition Node

Sets the relative position of an element from the center of the element's anchors.

**Node Inputs**

**Activate**
- Sets the relative position of the element.

**ElementName**
- Name of the element.

**ElementID**
- The element Id. Used if `ElementName` is empty.

**XPosition**
- The relative x position of the element.

**YPosition**
- The relative y position of the element.

**Node Outputs**

**Done**
- Sends a signal when the node's action is finished.

UIe:Transform:GetZRotation Node

Gets the z rotation of an element.
**Node Inputs**

**Activate**

Updates the output.

**ElementName**

Name of the element.

**ElementID**

The element Id. Used if `ElementName` is empty.

**Node Outputs**

**Value**

The z rotation of the element.

**UIe:Transform:SetZRotation Node**

Sets the z rotation of an element.

**Node Inputs**

**Activate**

Sets the z rotation of the element.

**ElementName**

Name of the element.

**ElementID**

The element Id. Used if `ElementName` is empty.

**Node Outputs**

**Done**

Sends a signal when the node's action is finished.

**UIe:Transform:MoveCanvasPositionBy Node**

Moves an element in canvas space.

**Node Inputs**

**Activate**

Moves the element.

**ElementName**

Name of the element.

**ElementID**

The element Id. Used if `ElementName` is empty.
**XOffset**

The x offset value.

**YOffset**

The y offset value.

**Node Outputs**

**Done**

Sends a signal when the node's action is finished.

**UIe:Transform:MoveLocalPositionBy Node**

Moves an element relative to the center of the element's anchors.

**Node Inputs**

**Activate**

Moves the element.

**ElementName**

Name of the element.

**ElementID**

The element Id. Used if ElementName is empty.

**XOffset**

The x offset value.

**YOffset**

The y offset value.

**Node Outputs**

**Done**

Sends a signal when the node's action is finished.

**UIe Animation Node**

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 about the latest features, see the Amazon Lumberyard User Guide.

The UI animation node consists of the following node inputs and outputs:

**UIe:Sequence:Play Node**

Controls playback of a UI animation sequence.
Node Inputs

Start
Starts playing the sequence from the beginning and triggers the OnStarted output.

Stop
Jumps the animation to the end and stops playing and triggers the OnStopped output.

Abort
Jumps the animation to the end and stops playing and triggers the OnAborted output.

Pause
Pauses the animation.

Resume
Continues playing a previously paused animation.

Reset
Resets the animation to the start. This applies all the key values for the first keyframe of the animation.

SequenceName
The name of the sequence to play.

Node Outputs

OnStarted
Triggers when the sequence starts playing.

OnStopped
Triggers an output when the sequence stops playing, either because the end of the animation is reached or because the sequence is forced to stop (for example, by using the Stop node input).

OnAborted
Triggers an output when the sequence is aborted (for example, by using the Abort node input).

UI Flow Graph Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

The UI flow graph node set is the original, and now legacy, version of the UI flow graph nodes.

Use the Ule (p. 483) flow graph node set for best results when creating new flow graph nodes for your user interface.
Topics

- UI Canvas Nodes (p. 561)
- UI Component Nodes (p. 566)
- UI Animation Node (p. 641)

UI Canvas Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

The UI Canvas flow graph nodes have been superseded by the UI Canvas (p. 483) flow graph nodes. For best results, use the UI Canvas (p. 483) flow graph nodes.

You can use these flow graph nodes to perform actions on a UI canvas.

**UI:Canvas:ActionListener Node**

Listens for the specified action on a UI canvas.

**Node Inputs**

**Activate**

Initiates listening for the specified action.

**CanvasID**

Unique ID of the canvas to listen to.

**ActionName**

Name of the action to listen for.

**Node Outputs**

**OnAction**

Triggers when the canvas sends the action.

**ElementID**

ID of the UI element that triggered the action.

**UI:Canvas:Load Node**

Loads the specified UI canvas.

**Node Inputs**

**Activate**

Loads the canvas.
CanvasPathname

Path of the canvas to load.

Disabled

Sets whether canvas is disabled initially. If disabled, the canvas is not updated or rendered.

Node Outputs

OnLoad

Sends a signal when the canvas is loaded.

CanvasID

Outputs the unique canvas ID when the canvas is loaded.

UI:Canvas:Unload Node
Unloads the specified canvas.

Node Inputs

Activate

Unloads the canvas.

CanvasID

Unique ID of the canvas to unload.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Canvas:FindLoaded Node
Finds the canvas ID for the UI canvas file path.

Node Inputs

Activate

Finds the canvas using the UI canvas file path.

CanvasPathname

Path of the canvas to find.

Node Outputs

CanvasID

The ID of the canvas that was found (if it was found).

Found

True if the canvas was found; otherwise, false.
UI:Canvas:GetKeepLoaded Node

Gets the Boolean value of whether the canvas stays loaded when a level is unloaded.

Node Inputs

Activate

Gets whether the canvas stays loaded when the level is unloaded.

CanvasID

Unique ID of the canvas to keep loaded.

Node Output

KeepLoaded

The Boolean value of whether the canvas stays loaded if the level is unloaded. True if the canvas should stay loaded during level unload; otherwise, false.

UI:Canvas:SetKeepLoaded Node

Determines whether the canvas stays loaded when a level is unloaded.

Node Inputs

Activate

Sets whether the canvas stays loaded when the level is unloaded.

CanvasID

Unique ID of the canvas to keep loaded.

KeepLoaded

If true, causes the canvas to stay loaded when the level is unloaded.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Canvas:GetDrawOrder Node

Gets the integer draw order value for a UI canvas with respect to other UI canvases.

Node Inputs

Activate

Gets the draw order for the canvas.

CanvasID

Unique ID of the canvas to get the draw order from.
Node Output

**DrawOrder**

Order in which the canvas draws. Higher numbers appear before lower numbers.

**UI:Canvas:SetDrawOrder Node**

Sets the draw order for a UI canvas with respect to other UI canvases.

**Node Inputs**

- **Activate**
  
  Sets the draw order for the canvas.

- **CanvasID**
  
  Unique ID of the canvas whose draw order you are setting.

- **DrawOrder**
  
  Order in which to display the canvas. Higher numbers appear before lower numbers.

**Node Output**

- **Done**
  
  Sends a signal when the node's action is finished.

**UI:Canvas:GetIsPixelAligned Node**

Gets the boolean value of whether the canvas is pixel-aligned.

**Node Inputs**

- **Activate**
  
  Gets whether visual element's vertices should snap to the nearest pixel.

- **CanvasID**
  
  Unique ID of the canvas.

**Node Output**

- **IsPixelAligned**
  
  Boolean value. True if the visual element's vertices should snap to the nearest pixel; otherwise, false.

**UI:Canvas:SetIsPixelAligned Node**

Sets whether visual element's vertices should snap to the nearest pixel.

**Node Inputs**

- **Activate**
  
  Sets the pixel-aligned property for the canvas ID.
CanvasID

Unique ID of the canvas to receive the pixel-aligned property value.

IsPixelAligned

Boolean value that represents whether a visual element's vertices should snap to the nearest pixel.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Canvas:GetEnabled Node

Gets the boolean enabled flag of the canvas. Enabled canvases are updated and each frame rendered.

Node Inputs

Activate

Gets the enabled flag of the canvas.

CanvasID

Unique ID of the canvas to obtain the enabled flag from.

Node Output

Enabled

The enabled flag of the canvas. True if enabled; otherwise, false.

UI:Canvas:SetEnabled Node

Sets whether the canvas is enabled. Enabled canvases are updated and each frame rendered.

Node Inputs

Activate

Sets the enabled flag of the canvas.

CanvasID

Unique ID of the canvas to obtain the enabled flag from.

Enabled

True if the canvas should be enabled; otherwise, false.

Node Output

Done

Sends a signal when the node's action is finished.
UI Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

The UI component flow graph nodes have been superseded by the UIe Component (p. 487) flow graph nodes. For best results, use the UIe Component (p. 487) flow graph nodes.

These flow graph nodes perform actions on UI elements through their components.

Topics
- UI Button Component Nodes (p. 566)
- UI Checkbox Component Nodes (p. 567)
- UI DynamicLayout Component Nodes (p. 573)
- UI DynamicScrollbox Component Nodes (p. 573)
- UI Element Node (p. 575)
- UI Fader Component Nodes (p. 577)
- UI Image Component Nodes (p. 579)
- UI Interactable Component Nodes (p. 583)
- UI Layout Column Component Nodes (p. 584)
- UI Layout Grid Component Nodes (p. 587)
- UI Layout Row Component Nodes (p. 593)
- UI Mask Component Nodes (p. 597)
- UI ScrollBox Component Nodes (p. 600)
- UI ScrollBar Component Nodes (p. 612)
- UI Slider Component Nodes (p. 618)
- UI Text Component Nodes (p. 626)
- UI Text Input Component Nodes (p. 631)

UI Button Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

The UI Button component flow graph nodes have been superseded by the UI Button (p. 488) flow graph nodes. For best results, use the UI Button (p. 488) component flow graph nodes.
Use the following flow graph nodes to perform actions on the button component.

**UI:Button:GetActionName Node**

Gets the action name string that is emitted when the button is released.

**Node Inputs**

- **Activate**
  Updates the output.

- **CanvasID**
  Unique identifier of the element's canvas.

- **ElementID**
  Unique identifier of the button element.

**Node Output**

- **Action**
  The action name associated with the button.

**UI:Button:SetActionName Node**

Sets the action name string that's emitted when the button is released.

**Node Inputs**

- **Activate**
  Assigns the action name.

- **CanvasID**
  Unique identifier of the element's canvas.

- **ElementID**
  Unique identifier of the button element.

- **Action**
  The action name string to assign to the button.

**Node Output**

- **Done**
  Sends a signal when the node's action is finished.

**UI Checkbox Component Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.
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 about the latest features, see the Amazon Lumberyard User Guide.

The UI component flow graph nodes have been superseded by the UIe Checkbox (p. 489) component flow graph nodes. For best results, use the UIe Checkbox (p. 489) component flow graph nodes.

Use the following flow graph nodes to perform actions on the check box component.

**UI:Checkbox:GetState Node**

Gets the Boolean state of the check box.

**Node Inputs**

**Activate**

Gets the state of the check box.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the check box element.

**Node Output**

**State**

Outputs the current Boolean state of the check box.

**UI:Checkbox:SetState Node**

Sets the Boolean state of the check box.

**Node Inputs**

**Activate**

Sets the state of the check box.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the check box element.

**Node Output**

**State**

The Boolean state of the check box.

**Done**

Sends a signal when the node's action is finished.
UI:Checkbox:GetChangedActionName Node

Gets the action triggered when the check box value changed.

**Node Inputs**

- **Activate**
  
  Gets the changed action name.

- **CanvasID**
  
  Unique identifier of the element's canvas.

- **ElementID**
  
  Unique identifier of the check box element.

**Node Output**

- **ChangedAction**
  
  The action name string value emitted when the check box value changes.

UI:Checkbox:SetChangedActionName Node

Sets the action triggered when the check box value changed.

**Node Inputs**

- **Activate**
  
  Gets the changed action name.

- **CanvasID**
  
  Unique identifier of the element's canvas.

- **ElementID**
  
  Unique identifier of the check box element.

- **ChangedAction**
  
  The action name string value emitted when the check box value changes.

UI:Checkbox:GetOptionalCheckedEntity Node

Gets the child element to show when the check box is in the on state.

**Node Inputs**

- **Activate**
  
  Updates the output.

- **CanvasID**
  
  Unique identifier of the element's canvas.

- **ElementID**
  
  Unique identifier of the check box element.
Node Output

CheckedElement

The child element to show when the check box is selected (in the \textit{on} state).

UI:Checkbox:SetOptionalCheckedEntity Node

Sets the child element to show when the check box is selected (in the \textit{on} state).

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the check box element.

CheckedElement

The child element to show when the check box is selected (in the \textit{on} state).

Node Output

Done

Sends a signal when the node's action is finished.

UI:Checkbox:GetOptionalUncheckedEntity Node

Gets the child element to show when the check box is deselected (in the \textit{off} state).

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the check box element.

UncheckedElement

The child element to show when the check box is deselected (off state).

UI:Checkbox:SetOptionalUncheckedEntity Node

Sets the child element to show when the check box is deselected (in the \textit{off} state).
Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the check box element.

UncheckedElement

The child element to show when the check box is deselected (in the off state).

Node Output

Done

Sends a signal when the node's action is finished.

UI:Checkbox:GetTurnOnActionName Node

Gets the action triggered when the check box is selected.

Node Inputs

Activate

Update the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the check box element.

Node Output

TurnOnAction

The action name emitted when the check box is selected (turned on).

UI:Checkbox:SetTurnOnActionName Node

Sets the action triggered when the check box is selected (turned on).

Node Inputs

Activate

Assigns TurnOnAction as the action name that is emitted when the check box is selected.

CanvasID

Unique identifier of the element's canvas.
ElementID
   Unique identifier of the check box element.

TurnOnAction
   The action name emitted when the check box is selected.

Node Output

Done
   Sends a signal when the node's action is finished.

UI:Checkbox:GetTurnOffActionName Node

Gets the action triggered when the check box is deselected (turned off).

Node Inputs

Activate
   Update the output.

CanvasID
   Unique identifier of the element's canvas.

ElementID
   Unique identifier of the check box element.

Node Output

TurnOffAction
   The action name emitted when the check box is deselected.

UI:Checkbox:SetTurnOffActionName Node

Sets the action triggered when the check box is deselected (turned off).

Node Inputs

Activate
   Assigns TurnOffAction as the action name that is emitted when the check box is deselected.

CanvasID
   Unique identifier of the element's canvas.

ElementID
   Unique identifier of the check box element.

TurnOffAction
   The action name emitted when the check box is deselected.
Node Output

Done

Sends a signal when the node's action is finished.

UI DynamicLayout Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

The UI DynamicLayout flow graph nodes have been superseded by the UI Dynamic Layout flow graph nodes. For best results, use the UI Dynamic Layout flow graph nodes.

Use the following flow graph nodes to perform actions on the dynamic layout component.

UI:DynamicLayout:SetNumChildElements Node

Sets the number of child elements of the layout element. The child elements are cloned from a prototype element.

Node Inputs

Activate

Sets the number of child elements.

ElementName

Name of the layout element.

ElementId

The element Id of the layout element. Used if ElementName is empty.

NumChildElements

The number of child elements.

Node Output

Done

Sends a signal when the node's action is finished.

UI DynamicScrollbox Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.
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 about the latest features, see the Amazon Lumberyard User Guide.

The **UI DynamicScrollBox** flow graph nodes have been superseded by the **UIe DynamicScrollBox** (p. 494) flow graph nodes. For best results, use the **UIe DynamicScrollBox** flow graph nodes.

Use the following flow graph nodes to perform actions on the dynamic scroll box component.

### UI:DynamicScrollBox:GetLocationIndexOfChild Node

Gets the index of the specified child of the dynamic scroll box's content element. Use this node rather than **UI:Element:GetIndexOfChild**, since dynamic scroll boxes create only the minimum number of children for display.

**Node Inputs**

- **Activate**
  - Updates the output.
- **ElementName**
  - Name of the dynamic scroll box element.
- **ElementId**
  - The element Id of the dynamic scroll box element. Used if **ElementName** is empty.
- **ChildElementId**
  - The element Id of the child.

**Node Output**

- **Index**
  - The index of the child.

### UI:DynamicScrollBox:RefreshContent Node

Refreshes the dynamic scroll box by retrieving the number of the content element's children, setting up the content element according to the number of children, and notifying listeners of the child elements that are visible.

**Node Inputs**

- **Activate**
  - Refreshes the content.
- **ElementName**
  - Name of the dynamic scroll box element.
- **ElementId**
  - The element Id of the dynamic scroll box element. Used if **ElementName** is empty.
Node Output

Done

Sends a signal when the node's action is finished.

UI Element Node

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

The UI Element flow graph nodes have been superseded by the UI Element (p. 495) flow graph nodes. For best results, use the UI Element (p. 495) flow graph nodes.

Use the following flow graph nodes to perform actions on an element.

UI:Element:IsEnabled Node

Gets whether the element is enabled.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

State

The enabled state of the element.

UI:Element:SetIsEnabled Node

Sets the Boolean enabled state of the element. If an element is not enabled, neither it nor any of its children are drawn or interactive.

Node Inputs

Activate

Sets the enabled state to the value of the State input.

CanvasID

Unique identifier of the element's canvas.
**ElementID**

Unique identifier of the element.

**State**

The Boolean enabled state of the element.

**Node Output**

**Done**

Sends a signal when the node's action is finished.

**UI:Element:GetChildAtIndex Node**

Gets the child of an element at the specified index.

**Node Inputs**

**Activate**

Updates the outputs.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**ChildIndex**

The index of the child.

**Node Output**

**ChildElementId**

The element Id of the child element.

**UI:Element:GetIndexOfChild Node**

Gets the index of the specified child.

**Node Inputs**

**Activate**

Updates the outputs.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**ChildIndex**

The index of the child.

**Node Output**

**ChildElementId**

The element Id of the child element.
**Node Output**

**IndexOfChild**

The index of the child element.

**UI:Element:GetNumChildElements Node**

Gets the number of children of an element.

**Node Inputs**

**Activate**

Updates the outputs.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**Node Output**

**NumChildElements**

The number of child elements.

**UI:Element:GetParent Node**

Gets the parent of an element.

**Node Inputs**

**Activate**

Updates the outputs.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**Node Output**

**ParentElementId**

The element Id of the parent element.

**UI Fader Component Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about [Script Canvas](https://www.lumberyardengine.com/tutorials/script-canvas), Lumberyard's new visual scripting environment.
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 about the latest features, see the Amazon Lumberyard User Guide.

The UI Fader component flow graph nodes have been superseded by the UIe Fader (p. 499) component flow graph nodes. For best results, use the UIe Fader (p. 499) component flow graph nodes.

Use the following flow graph nodes to perform actions on the fader component.

**UI:Fader:Animation Node**

Animates the fader component on the specified element.

**Node Inputs**

- **Activate**
  - Starts a fade animation.
- **CanvasID**
  - Unique identifier of the fader element's canvas.
- **ElementID**
  - Unique identifier of the fader element.
- **StartValue**
  - Value at which the fade starts.
  - Valid values: 0 = Invisible | 1 = Opaque | -1 = Start from the current value
- **TargetValue**
  - Value at which the fade ends.
  - Valid values: 0 = Invisible | 1 = Opaque
- **Speed**
  - Rate at which the element fades.
  - Valid values: 0 = Instant fade | 0.5 = Slow fade | 1 = One second fade | 2 = Fade twice as fast

**Node Outputs**

- **OnComplete**
  - Sends a signal when the fade action is finished.
- **OnInterrupted**
  - Sends a signal when the fade is interrupted by another fade starting.

**UI:Fader:GetFadeValue Node**

Gets the floating-point fade value of an element.
Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Value

The floating-point fade value of the element (ElementID).

UI:Fader:SetFadeValue Node

Sets the fade value of an element.

Node Inputs

Activate

When triggered, assigns Value as the fade value of the fader component of the element.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Value

The fade value to assign to the fader component for the element.

Node Output

Done

Sends a signal when the node's action is finished.

UI Image Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.
The UI Image component flow graph nodes have been superseded by the UIe Image (p. 501) component flow graph nodes. For best results, use the UIe Image (p. 501) component flow graph nodes.

Use the following flow graph nodes to perform actions on the image component.

**UI:Image:GetImageSource Node**


Retrieves the texture file path currently used by the specified image element.

**Node Inputs**

- **Activate**
  - Updates the output.
- **CanvasID**
  - Unique identifier of the element's canvas.
- **ElementID**
  - Unique identifier of the image element.

**Node Outputs**

- **Value**
  - Outputs the file path of the image that is currently on the element.

**UI:Image:SetImageSource Node**


Changes the texture on the specified image element.

**Node Inputs**

- **Activate**
  - Set the texture.
- **CanvasID**
  - Unique identifier of the element's canvas.
- **ElementID**
  - Unique identifier of the image element.
- **ImagePath**
  - File path of the texture to display.

**Node Output**

- **Done**
  - Sends a signal when the node's action is finished.
**UI:Image:GetSprite Node**

Gets the texture file path currently used by the specified image element.

**Node Inputs**

- **Activate**
  - Updates the output.

- **CanvasID**
  - Unique ID of the element's canvas.

- **ElementID**
  - Unique ID of the image element.

**Node Output**

- **Value**
  - Outputs the file path of the image that is currently on the element.

**UI:Image:SetSprite Node**

Sets the texture on the specified image element.

**Node Inputs**

- **Activate**
  - Sets the texture.

- **CanvasID**
  - Unique ID of the element's canvas.

- **ElementID**
  - Unique ID of the image element.

- **ImagePath**
  - File path of the texture to display.

**Node Output**

- **Done**
  - Sends a signal when the node's action is finished.

**UI:Image:GetImageType Node**

Gets the type of the image. Affects how the texture or sprite is mapped to the image rectangle.

**Node Inputs**

- **Activate**
  - Updates the output.
CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the image element.

Node Output

ImageType

An integer representing how the image is scaled and placed.

Valid values: 0 = Stretched | 1 = Sliced | 2 = Fixed | 3 = Tiled | 4 = Stretched to fit | 5 = Stretched to fill

UI:Image:SetImageType Node

Sets the type of the image. Affects how the texture or sprite is mapped to the image rectangle.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the image element.

ImageType

An integer representing how the image is scaled and placed.

Valid values: 0 = Stretched | 1 = Sliced | 2 = Fixed | 3 = Tiled | 4 = Stretched to fit | 5 = Stretched to fill

Node Output

Done

Sends a signal when the node's action is finished.

UI:Image:GetColor Node

Gets the color tint for the image.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.
ElementID

Unique identifier of the image element.

Node Outputs

Color

The RGB value (0 – 255 each for R, G, and B) of the element (ElementID).

Alpha

The alpha value (0 – 255) of the element (ElementID).

UI:Image:SetColor Node

Sets the color tint for the image.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the image element.

Color

The RGB value (0 – 255 each for R, G, and B).

Alpha

A floating-point alpha value (0 – 255).

Node Output

Done

Sends a signal when the node's action is finished.

UI Interactable Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

The UI Interactable component flow graph nodes have been superseded by the UI Interactable (p. 504) component flow graph nodes. For best results, use the UI Interactable (p. 504) component flow graph nodes.
Use the following flow graph node for the **Interactable** component.

**UI:Interactable:SetIsHandlingEvents Node**

Sets the Boolean "is handling events" state of the element.

The **Interactable** flow graph nodes can be used to get or set values on any interactive UI element.

Interactive UI elements are elements that players can interact with in game, such as button, text input, check box, slider, and so on. The **SetIsHandlingEvents** flow graph node sets whether an interactive UI element should handle input events. If set to false, then the UI element does not respond to input events, and its visual state is also changed to disabled.

**Node Inputs**

- **Activate**
  - Sets the "is handling events" state.

- **CanvasID**
  - Unique identifier of the element's canvas.

- **ElementID**
  - Unique identifier of the element.

- **State**
  - The Boolean "is handling events" state of the element.

**Node Output**

- **Done**
  - Sends a signal when the node's action is finished.

**UI Layout Column Component Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about **Script Canvas**, Lumberyard's new visual scripting environment.

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 about the latest features, see the **Amazon Lumberyard User Guide**.

The UI Layout column component flow graph nodes have been superseded by the **UIe Layout column** (p. 505) component flow graph nodes. For best results, use the **UIe Layout column** (p. 505) component flow graph nodes.

Use the following flow graph nodes to perform actions on the layout column component.

**UI:LayoutColumn:GetOrder Node**

Gets the vertical order of the **LayoutColumn** component for an element.
**Node Inputs**

**Activate**
- Updates the output.

**CanvasID**
- Unique identifier of the element's canvas.

**ElementID**
- Unique identifier of the element.

**Node Output**

**Order**
- An integer representing the vertical order.
  - Valid values: 0 = Top to bottom | 1 = Bottom to top

**UI:LayoutColumn:SetOrder Node**
Sets the vertical order of the LayoutColumn component for an element.

**Node Inputs**

**Activate**
- Sets the vertical order for the element.

**CanvasID**
- Unique identifier of the element's canvas.

**ElementID**
- Unique identifier of the element.

**Order**
- An integer representing the vertical order. 0 = Top to bottom | 1 = Bottom to top.

**Node Output**

**Done**
- Sends a signal when the node's action is finished.

**UI:LayoutColumn:GetPadding Node**
Gets the padding (in pixels) inside the edges of the LayoutColumn component for an element.

**Node Inputs**

**Activate**
- Updates the outputs.

**CanvasID**
- Unique identifier of the element's canvas.
**ElementID**

Unique identifier of the element.

**Node Outputs**

**Left**

An integer representing the padding inside the left edge of the element.

**Right**

An integer representing the padding inside the right edge of the element.

**Top**

An integer representing the padding inside the top edge of the element.

**Bottom**

An integer representing the padding inside the bottom edge of the element.

**UI:LayoutColumn:SetPadding Node**

Sets the padding (in pixels) inside the edges of the LayoutColumn component for an element.

**Node Inputs**

**Activate**

Updates the outputs.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**Left**

An integer representing the padding inside the left edge of the element.

**Right**

An integer representing the padding inside the right edge of the element.

**Top**

An integer representing the padding inside the top edge of the element.

**Bottom**

An integer representing the padding inside the bottom edge of the element.

**Node Output**

**Done**

Sends a signal when the node's action is finished.

**UI:LayoutColumn:GetSpacing Node**

Gets the spacing (in pixels) between child elements of the LayoutColumn component for an element.
Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Spacing

A float value of the spacing (in pixels) between child elements of the element (ElementID).

UI:LayoutColumn:SetSpacing Node

Sets the spacing (in pixels) between child elements of the LayoutColumn component for an element.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Spacing

A float value of the spacing (in pixels) between child elements of the element (ElementID).

Node Output

Done

Sends a signal when the node's action is finished.

UI Layout Grid Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.
The UI Layout grid component flow graph nodes have been superseded by the UI Layout grid component flow graph nodes. For best results, use the UI Layout grid component flow graph nodes.

Use the following flow graph nodes to perform actions on the layout grid component.

**UI:LayoutGrid:GetCellSize Node**

Gets the size (in pixels) of a child element in the layout.

**Node Inputs**

- **Activate**
  - Updates the output.

- **CanvasID**
  - Unique identifier of the element's canvas.

- **ElementID**
  - Unique identifier of the element.

**Node Outputs**

- **CellWidth**
  - The width (in pixels) of a child element of element (ElementID).

- **CellHeight**
  - The height (in pixels) of a child element of element (ElementID).

**UI:LayoutGrid:SetCellSize Node**

Sets the size (in pixels) of a child element in the layout.

**Node Inputs**

- **Activate**
  - Updates the output.

- **CanvasID**
  - Unique identifier of the element's canvas.

- **ElementID**
  - Unique identifier of the element.

- **CellWidth**
  - The width (in pixels) of a child element of element (ElementID).

- **CellHeight**
  - The height (in pixels) of a child element of element (ElementID).

**Node Output**

- **Done**
  - Sends a signal when the node's action is finished.
UI:LayoutGrid:GetHorizontalOrder Node

Gets the horizontal order for the layout.

**Node Inputs**

- **Activate**
  - Updates the outputs.
- **CanvasID**
  - Unique identifier of the element's canvas.
- **ElementID**
  - Unique identifier of the element.

**Node Output**

- **Order**
  - An integer representing the horizontal order.
  - Valid values: 0 = Left to right | 1 = Right to left

UI:LayoutGrid:SetHorizontalOrder Node

Sets the horizontal order for the layout.

**Node Inputs**

- **Activate**
  - Updates the outputs.
- **CanvasID**
  - Unique identifier of the element's canvas.
- **ElementID**
  - Unique identifier of the element.
- **Order**
  - An integer representing the horizontal order.
  - Valid values: 0 = Left to right | 1 = Right to left

**Node Output**

- **Done**
  - Sends a signal when the node's action is finished.

UI:LayoutGrid:GetPadding Node

Gets the padding (in pixels) inside the edges of the LayoutGrid component for an element.
Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Left

An integer representing the padding inside the left edge of the element.

Right

An integer representing the padding inside the right edge of the element.

Top

An integer representing the padding inside the top edge of the element.

Bottom

An integer representing the padding inside the bottom edge of the element.

UI:LayoutGrid:SetPadding Node

Sets the padding (in pixels) inside the edges of the LayoutGrid component for an element.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Left

An integer representing the padding inside the left edge of the element.

Right

An integer representing the padding inside the right edge of the element.

Top

An integer representing the padding inside the top edge of the element.

Bottom

An integer representing the padding inside the bottom edge of the element.
Node Output

Done

Sends a signal when the node's action is finished.

**UI:LayoutGrid:GetSpacing Node**

Gets the spacing (in pixels) between child elements of the LayoutGrid component for an element.

**Node Inputs**

- **Activate**
  
  Updates the output.

- **CanvasID**
  
  Unique identifier of the element's canvas.

- **ElementID**
  
  Unique identifier of the element.

**Node Output**

- **Spacing**
  
  A float value of the spacing (in pixels) between child elements of the element (*ElementID*).

**UI:LayoutGrid:SetSpacing Node**

Sets the spacing (in pixels) between child elements of the LayoutGrid component for an element.

**Node Inputs**

- **Activate**
  
  Updates the output.

- **CanvasID**
  
  Unique identifier of the element's canvas.

- **ElementID**
  
  Unique identifier of the element.

- **Spacing**
  
  A float value of the spacing (in pixels) between child elements of the element (*ElementID*).

**Node Output**

- **Done**
  
  Sends a signal when the node's action is finished.

**UI:LayoutGrid:GetStartingDirection Node**

Gets the starting direction for the layout.
Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Direction

An integer representing the direction.

Valid values: 0 = Horizontal order | 1 = Vertical order

UI:LayoutGrid:SetStartingDirection Node

Sets the starting direction for the layout.

Node Inputs

Activate

Set the starting direction for the layout.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Direction

An integer representing the horizontal order.

Valid values: 0 = Horizontal order | 1 = Vertical order.

Node Output

Done

Sends a signal when the node's action is finished.

UI:LayoutGrid:GetVerticalOrder Node

Gets the vertical order for the layout.

Node Inputs

Activate

Updates the outputs.
CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Action

An integer representing the vertical order.

Valid values: 0 = Top to bottom | 1 = Bottom to top

UI:LayoutGrid:SetVerticalOrder Node

Sets the vertical order for the layout.

Node Inputs

Activate

Sets the vertical order for the layout.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Action

An integer representing the vertical order.

Valid values: 0 = Top to bottom | 1 = Bottom to top

Node Output

Done

Sends a signal when the node's action is finished.

UI Layout Row Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.
The UI Layout row component flow graph nodes have been superseded by the UIe Layout row (p. 513) component flow graph nodes. For best results, use the UIe Layout row (p. 513) component flow graph nodes.

Use the following flow graph nodes to perform actions on the layout row component.

**UI:LayoutRow:GetEnumerator Node**

Gets the horizontal order of the LayoutRow component for an element.

**Node Inputs**

**Activate**

Updates the output.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**Node Output**

**Order**

An integer representing the horizontal order.

Valid values: 0 = Left to right | 1 = Right to left

**UI:LayoutRow:SetOrder Node**

Sets the horizontal order of the LayoutRow component for an element.

**Node Inputs**

**Activate**

Updates the output.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**Order**

An integer representing the horizontal order.

Valid values: 0 = Left to right | 1 = Right to left

**Node Output**

**Done**

Sends a signal when the node's action is finished.
UI:LayoutRow:GetPadding Node

Gets the padding (in pixels) inside the edges of the LayoutRow component for an element.

**Node Inputs**

**Activate**

Updates the output.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**Node Outputs**

**Left**

An integer representing the padding inside the left edge of the element.

**Right**

An integer representing the padding inside the right edge of the element.

**Top**

An integer representing the padding inside the top edge of the element.

**Bottom**

An integer representing the padding inside the bottom edge of the element.

UI:LayoutRow:SetPadding Node

Sets the padding (in pixels) inside the edges of the LayoutRow component for an element.

**Node Inputs**

**Activate**

Sets the padding (in pixels) inside the edges of the LayoutRow.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**Left**

An integer representing the padding inside the left edge of the element.

**Right**

An integer representing the padding inside the right edge of the element.

**Top**

An integer representing the padding inside the top edge of the element.
**Bottom**

An integer representing the padding inside the bottom edge of the element.

**Node Output**

**Done**

Sends a signal when the node’s action is finished.

**UI:LayoutRow:GetSpacing Node**

Gets the spacing (in pixels) between child elements of the LayoutRow component for an element.

**Node Inputs**

**Activate**

Updates the outputs.

**CanvasID**

Unique identifier of the element’s canvas.

**ElementID**

Unique identifier of the element.

**Node Output**

**Spacing**

A float value of the spacing (in pixels) between child elements of the element (*ElementID*).

**UI:LayoutRow:SetSpacing Node**

Sets the spacing (in pixels) between child elements of the LayoutRow component for an element.

**Node Inputs**

**Activate**

Sets the spacing (in pixels) between child elements.

**CanvasID**

Unique identifier of the element’s canvas.

**ElementID**

Unique identifier of the element.

**Spacing**

A float value of the spacing (in pixels) between child elements of the element (*ElementID*).

**Node Output**

**Done**

Sends a signal when the node’s action is finished.
UI Mask Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

The UI Mask component flow graph nodes have been superseded by the UIe Mask (p. 516) component flow graph nodes. For best results, use the UIe Mask (p. 516) component flow graph nodes.

Use the following flow graph nodes to perform actions on the mask component.

**UI:Mask:GetDrawBehind Node**

Gets whether mask is drawn behind the child elements.

**Node Inputs**

- **Activate**
  
  Updates the outputs.

- **CanvasID**

  Unique identifier of the element's canvas.

- **ElementID**

  Unique identifier of the element.

**Node Output**

**DrawBehind**

Indicates whether mask is drawn behind the child elements.

**UI:Mask:SetDrawBehind Node**

Sets whether mask is drawn behind the child elements.

**Node Inputs**

- **Activate**

  Sets whether mask is drawn behind the child elements.

- **CanvasID**

  Unique identifier of the element's canvas.

- **ElementID**

  Unique identifier of the element.
**DrawBehind**

Sets whether mask is drawn behind the child elements.

**Node Output**

**Done**

Sends a signal when the node’s action is finished.

**UI:Mask:GetDrawInFront Node**

Gets whether mask is drawn in front of child elements.

**Node Inputs**

**Activate**

Updates the outputs.

**CanvasID**

Unique identifier of the element’s canvas.

**ElementID**

Unique identifier of the element.

**Node Output**

**DrawInFront**

Indicates whether mask is drawn in front of child elements.

**UI:Mask:SetDrawInFront Node**

Sets whether mask is drawn in front of child elements.

**Node Inputs**

**Activate**

Sets whether mask is drawn in front of child elements.

**CanvasID**

Unique identifier of the element’s canvas.

**ElementID**

Unique identifier of the element.

**DrawInFront**

Sets whether mask is drawn in front of child elements.

**Node Output**

**Done**

Sends a signal when the node’s action is finished.
UI:Mask:GetIsMaskingEnabled Node

Gets whether masking is enabled.

**Node Inputs**

**Activate**

Updates the outputs.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**Node Output**

**IsMaskingEnabled**

Indicates whether masking is enabled.

---

UI:Mask:SetIsMaskingEnabled Node

Sets whether masking is enabled.

**Node Inputs**

**Activate**

Sets whether masking is enabled.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**IsMaskingEnabled**

Sets whether masking is enabled.

**Node Output**

**Done**

Sends a signal when the node's action is finished.

---

UI:Mask:GetUseAlphaTest Node

Gets whether to use the alpha channel in the mask visual's texture to define the mask.

**Node Inputs**

**Activate**

Updates the outputs.
CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

UseAlphaTest

Indicates whether to use the alpha channel in the mask visual's texture to define the mask.

UI:Mask:SetUseAlphaTest Node

Sets whether to use the alpha channel in the mask visual's texture to define the mask.

Node Inputs

Activate

Sets whether to use the alpha channel in the mask visual's texture to define the mask.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

UseAlphaTest

Sets whether to use the alpha channel in the mask visual's texture to define the mask.

Node Output

Done

Sends a signal when the node's action is finished.

UI ScrollBox Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

The UI Scrollbox component flow graph nodes have been superseded by the UIe Scrollbox (p. 519) component flow graph nodes. For best results, use the UIe Scrollbox (p. 519) component flow graph nodes.
Use the following flow graph nodes to perform actions on the **ScrollBox** component.

**UI:ScrollBox:FindClosestContentChildElement Node**

Finds the child of the content element that is closest to the content anchors.

**Node Inputs**

- **Activate**
  
  Updates the outputs.

- **CanvasID**
  
  Unique identifier of the element's canvas.

- **ElementID**
  
  Unique identifier of the element.

**Node Output**

- **ClosestElement**
  
  The element currently closest to the focused element.

**UI:ScrollBox:GetContentEntity Node**

Gets the content element for the **ScrollBox**.

**Node Inputs**

- **Activate**
  
  Updates the outputs.

- **CanvasID**
  
  Unique identifier of the element's canvas.

- **ElementID**
  
  Unique identifier of the element.

**Node Output**

- **Content**
  
  The element that the **ScrollBox** scrolls.

**UI:ScrollBox:SetContentEntity Node**

Sets the content element for the **ScrollBox**.

**Node Inputs**

- **Activate**
  
  Sets the content element for the **ScrollBox**.
CanvasID
Unique identifier of the element's canvas.

ElementID
Unique identifier of the element.

Content
The element that the ScrollBox scrolls.

Node Output

Done
Sends a signal when the node's action is finished.

UI:ScrollBox:GetIsHorizontalScrollingEnabled Node
Gets whether the ScrollBox allows horizontal scrolling.

Node Inputs

Activate
Updates the outputs.

CanvasID
Unique identifier of the element's canvas.

ElementID
Unique identifier of the element.

Node Output

Enabled
Indicates whether horizontal scrolling is enabled.

UI:ScrollBox:SetIsHorizontalScrollingEnabled Node
Sets whether the ScrollBox allows horizontal scrolling.

Node Inputs

Activate
Sets whether the ScrollBox allows horizontal scrolling.

CanvasID
Unique identifier of the element's canvas.

ElementID
Unique identifier of the element.

Enabled
Sets whether horizontal scrolling is enabled.
Node Output

Done

Sends a signal when the node's action is finished.

UI:ScrollBox:GetIsScrollingConstrained Node

Gets whether the **ScrollBox** restricts scrolling to the content area.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

IsConstrained

Indicates whether scrolling is constrained.

UI:ScrollBox:SetIsScrollingConstrained Node

Sets whether the **ScrollBox** restricts scrolling to the content area.

Node Inputs

Activate

Sets whether the **ScrollBox** restricts scrolling to the content area.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

IsConstrained

Sets whether scrolling is constrained.

Node Output

Done

Sends a signal when the node's action is finished.

UI:ScrollBox:GetIsVerticalScrollingEnabled Node

Gets whether the **ScrollBox** allows vertical scrolling.
Node Inputs

Activate

   Updates the output.

CanvasID

   Unique identifier of the element's canvas.

ElementID

   Unique identifier of the element.

Node Output

Enabled

   Indicates whether vertical scrolling is enabled.

UI:ScrollBox:SetIsVerticalScrollingEnabled Node

Sets whether the ScrollBox allows vertical scrolling.

Node Inputs

Activate

   Sets whether the ScrollBox allows vertical scrolling.

CanvasID

   Unique identifier of the element's canvas.

ElementID

   Unique identifier of the element.

Enabled

   Sets whether vertical scrolling is enabled.

Node Output

Done

   Sends a signal when the node's action is finished.

UI:ScrollBox:GetScrollOffset Node

Gets the scroll offset of the ScrollBox.

Node Inputs

Activate

   Updates the outputs.

CanvasID

   Unique identifier of the element's canvas.
ElementID
Unique identifier of the element.

**Node Output**

**HorizOffset**
The horizontal scroll offset of the element identified by *ElementID*.

**VertOffset**
The vertical scroll offset of the element identified by *ElementID*.

**UI:ScrollBox:SetScrollOffset Node**
Sets the scroll offset of the *ScrollBox*.

**Node Inputs**

**Activate**
Sets the scroll offset of the *ScrollBox*.

**CanvasID**
Unique identifier of the element's canvas.

**ElementID**
Unique identifier of the element.

**HorizOffset**
The horizontal scroll offset of *ElementID*.

**VertOffset**
The vertical scroll offset of *ElementID*.

**Node Output**

**Done**
Sends a signal when the node's action is finished.

**UI:ScrollBox:GetScrollOffsetChangedActionName Node**
Gets the action triggered when the *ScrollBox* drag is completed.

**Node Inputs**

**Activate**
Updates the output.

**CanvasID**
Unique identifier of the element's canvas.

**ElementID**
Unique identifier of the element.
Node Output

ChangedAction

The action name.

UI:ScrollBox:SetScrollOffsetChangedActionName Node

Sets the action triggered when the ScrollBox drag is completed.

Node Inputs

Activate

Sets the action triggered when the ScrollBox drag is completed.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

ChangedAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

UI:ScrollBox:GetScrollOffsetChangingActionName Node

Gets the action triggered while the ScrollBox is being dragged.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

ChangingAction

The action name.

UI:ScrollBox:SetScrollOffsetChangingActionName Node

Sets the action triggered while the ScrollBox is being dragged.
**Node Inputs**

**Activate**
Sets the action triggered while the **ScrollBox** is being dragged.

**CanvasID**
Unique identifier of the element's canvas.

**ElementID**
Unique identifier of the element.

**ChangingAction**
The action name.

**Node Output**

**Done**
Sends a signal when the node's action is finished.

**UI:ScrollBox:GetSnapGrid Node**

Gets the snapping grid of the **ScrollBox**.

**Node Inputs**

**Activate**
Updates the outputs.

**CanvasID**
Unique identifier of the element's canvas.

**ElementID**
Unique identifier of the element.

**Node Outputs**

**HorizSpacing**
The horizontal grid spacing of the element identified by **ElementID**.

**VertSpacing**
The vertical grid spacing of the element identified by **ElementID**.

**UI:ScrollBox:SetSnapGrid Node**

Sets the snapping grid of the **ScrollBox**.

**Node Inputs**

**Activate**
Sets the snapping grid of the **ScrollBox**.
CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

HorizSpacing

The horizontal grid spacing of the element identified by ElementID.

VertSpacing

The vertical grid spacing of the element identified by ElementID.

Node Output

Done

Sends a signal when the node's action is finished.

UI:ScrollBox:GetSnapMode Node

Gets the snap mode for the ScrollBox.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

SnapMode

An integer representing the snap mode state.

Valid values: 0 = None | 1 = Children | 2 = Grid

UI:ScrollBox:SetSnapMode Node

Sets the snap mode for the ScrollBox.

Node Inputs

Activate

Sets the snap mode for the ScrollBox.

CanvasID

Unique identifier of the element's canvas.
ElementID
  Unique identifier of the element.

SnapMode
  An integer representing the snap mode state.
  Valid values: 0 = None | 1 = Children | 2 = Grid

Node Output
Done
  Sends a signal when the node's action is finished.

UI:ScrollBox:GetHorizontalScrollBarVisibility Node
Gets horizontal scroll bar visibility behavior.

Node Inputs
Activate
  Updates the outputs.

CanvasID
  Unique identifier of the element's canvas.

ElementID
  Unique identifier of the element.

Node Output
ScrollBarVisibility
  An integer that represents the scroll bar visibility behavior.
  Valid values: 0 = AlwaysVisible | 1 = AutoHide | 2 = AutoHideAndResizeViewArea

UI:ScrollBox:SetHorizontalScrollBarVisibility Node
Sets horizontal scroll bar visibility behavior.

Node Inputs
Activate
  Sets horizontal scroll bar visibility behavior.

CanvasID
  Unique identifier of the element's canvas.

ElementID
  Unique identifier of the element.
ScrollBarVisibility

An integer representing the scroll bar visibility behavior.

0 = AlwaysVisible | 1 = AutoHide | 2 = AutoHideAndResizeViewArea.

Node Output

Done

Sends a signal when the node's action is finished.

UI:ScrollBox:GetVerticalScrollBarVisibility Node

Gets vertical scroll bar visibility behavior.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

ScrollBarVisibility

An integer that represents the scroll bar visibility behavior.

Valid values: 0 = AlwaysVisible | 1 = AutoHide | 2 = AutoHideAndResizeViewArea

UI:ScrollBox:SetVerticalScrollBarVisibility Node

Sets vertical scroll bar visibility behavior.

Node Inputs

Activate

Sets vertical scroll bar visibility behavior.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

ScrollBarVisibility

An integer representing the scroll bar visibility behavior.

0 = AlwaysVisible | 1 = AutoHide | 2 = AutoHideAndResizeViewArea.
**Node Output**

**Done**

Sends a signal when the node's action is finished.

**UI:ScrollBox:GetHorizontalScrollBarEntity Node**

Gets the horizontal scroll bar element for the **ScrollBox**.

**Node Inputs**

**Activate**

Updates the outputs.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**Node Output**

**HorizontalScrollBar**

The element that scrolls the **ScrollBox** horizontally.

**UI:ScrollBox:SetHorizontalScrollBarEntity Node**

Sets the horizontal scroll bar element for the **ScrollBox**.

**Node Inputs**

**Activate**

Sets the horizontal scroll bar element for the **ScrollBox**.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**HorizontalScrollBar**

The element that scrolls the **ScrollBox** horizontally.

**Node Output**

**Done**

Sends a signal when the node's action is finished.

**UI:ScrollBox:GetVerticalScrollBarEntity Node**

Gets the vertical scroll bar element for the **ScrollBox**.
Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

VerticalScrollBar

The element that scrolls the ScrollBox vertically.

UI:ScrollBox:SetVerticalScrollBarEntity Node

Sets the vertical scroll bar element for the ScrollBox.

Node Inputs

Activate

Sets the vertical scroll bar element for the ScrollBox.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

VerticalScrollBar

The element that scrolls the ScrollBox vertically.

Node Output

Done

Sends a signal when the node's action is finished.

UI ScrollBar Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.
The UI ScrollBar component flow graph nodes have been superseded by the UI ScrollBar (p. 530) component flow graph nodes. For best results, use the UI ScrollBar (p. 530) component flow graph nodes.

Use the following flow graph nodes to perform actions on the ScrollBar component.

**UI:Scrollbar:GetHandleEntity Node**

Gets the handle element of the scroll bar.

**Node Inputs**

- **Activate**
  - Updates the output.
- **CanvasID**
  - Unique identifier of the element's canvas.
- **ElementID**
  - Unique identifier of the element.

**Node Output**

- **Handle**
  - The handle element.

**UI:Scrollbar:SetHandleEntity Node**

Sets the handle element of the scroll bar.

**Node Inputs**

- **Activate**
  - Sets the handle element.
- **CanvasID**
  - Unique identifier of the element's canvas.
- **ElementID**
  - Unique identifier of the element.
- **Handle**
  - The handle element.

**Node Output**

- **Done**
  - Sends a signal when the node's action is finished.

**UI:Scrollbar:GetValue Node**

Gets the value of the scroll bar.
Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Value

The scroll bar value of the element identified by ElementID.

UI:Scrollbar:SetValue Node

Sets the value of the scroll bar.

Node Inputs

Activate

Sets the value of the scroll bar.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Value

The scroll bar value of the element identified by ElementID.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Scrollbar:GetHandleSize Node

Gets the size of the handle.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.
**ElementID**

Unique identifier of the element.

**Node Output**

**HandleSize**

The size of the handle of the element identified by `ElementID`.

**UI:Scrollbar:SetHandleSize Node**

Sets the size of the handle.

**Node Inputs**

**Activate**

Sets the size of the handle.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**HandleSize**

The size of the handle of the element identified by `ElementID`.

**Node Output**

**Done**

Sends a signal when the node's action is finished.

**UI:Scrollbar:GetMinHandlePixelSize Node**

Gets the minimum size in pixels of the handle.

**Node Inputs**

**Activate**

Updates the output.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**HandleSize**

The size of the handle of the element identified by `ElementID`.

**Node Output**

**MinHandleSize**

The minimum size in pixels of the handle of the element identified by `ElementID`. 
UI:Scrollbar:SetMinHandlePixelSize Node
Sets the minimum size in pixels of the handle.

Node Inputs
Activate
Sets the minimum size in pixels of the handle.
CanvasID
Unique identifier of the element's canvas.
ElementID
Unique identifier of the element.
MinHandleSize
The minimum size in pixels of the handle of the element identified by ElementID.

Node Output
Done
Sends a signal when the node's action is finished.

UI:Scrollbar:GetValueChangedActionName Node
Gets the action triggered when the value is done changing.

Node Inputs
Activate
Updates the output.
CanvasID
Unique identifier of the element's canvas.
ElementID
Unique identifier of the element.

Node Output
ValueChangedAction
The action name.

UI:Scrollbar:SetValueChangedActionName Node
Sets the action triggered when the value is done changing.

Node Inputs
Activate
Updates the output.
CanvasID
Unique identifier of the element's canvas.

ElementID
Unique identifier of the element.

ValueChangedAction
The action name.

Node Output

Done
Sends a signal when the node's action is finished.

UI:Scrollbar:GetValueChangingActionName Node
Gets the action triggered while the value is changing.

Node Inputs

Activate
Updates the output.

CanvasID
Unique identifier of the element's canvas.

ElementID
Unique identifier of the element.

ValueChangingAction
The action name.

UI:Scrollbar:SetValueChangingActionName Node
Sets the action triggered while the value is changing.

Node Inputs

Activate
Updates the output.

CanvasID
Unique identifier of the element's canvas.

ElementID
Unique identifier of the element.

ValueChangingAction
The action name.
**Node Output**

**Done**

Sends a signal when the node's action is finished.

**UI Slider Component Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about [Script Canvas](https://docs.aws.amazon.com/lumberyard/latest/ug/script-canvas.html), Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

The **UI Slider** component flow graph nodes have been superseded by the **UIe Slider** component flow graph nodes. For best results, use the **UIe Slider** component flow graph nodes.

Use the following flow graph nodes to perform actions on the slider component.

**UI:Slider:GetFillEntity Node**

Gets the fill element.

**Node Inputs**

**Activate**

Updates the output.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**Node Output**

**FillElement**

The fill element.

**UI:Slider:SetFillEntity Node**

Sets the fill element.

**Node Inputs**

**Activate**

Sets the fill element.
CanvasID
Unique identifier of the element's canvas.

ElementID
Unique identifier of the element.

FillElement
The fill element.

Node Output

Done
Sends a signal when the node's action is finished.

UI:Slider:GetManipulatorEntity Node
Gets the manipulator element.

Node Inputs

Activate
Updates the output.

CanvasID
Unique identifier of the element's canvas.

ElementID
Unique identifier of the element.

Node Output

ManipulatorElement
The manipulator element.

UI:Slider:SetManipulatorEntity Node
Sets the manipulator element.

Node Inputs

Activate
Sets the manipulator element.

CanvasID
Unique identifier of the element's canvas.

ElementID
Unique identifier of the element.
**ManipulatorElement**

The manipulator element.

**Node Output**

**Done**

Sends a signal when the node's action is finished.

**UI:Slider:GetMaxValue Node**

Gets the maximum value of the slider.

**Node Inputs**

**Activate**

Updates the output.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**Node Output**

**MaxValue**

The slider maximum value of the element identified by **ElementID**.

**UI:Slider:SetMaxValue Node**

Sets the maximum value of the slider.

**Node Inputs**

**Activate**

Sets the maximum value of the slider.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**MaxValue**

The slider maximum value of the element identified by **ElementID**.

**Node Output**

**Done**

Sends a signal when the node's action is finished.
UI:Slider:GetMinValue Node

Gets the minimum value of the slider.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

MinValue

The slider minimum value of the element identified by ElementID.

UI:Slider:SetMinValue Node

Sets the minimum value of the slider.

Node Inputs

Activate

Sets the minimum value of the slider.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

MinValue

The slider minimum value of the element identified by ElementID.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Slider:GetStepValue Node

Gets the smallest increment allowed between values. Zero means no restriction.

Node Inputs

Activate

Updates the output.
Lumberyard Legacy Reference
UI Flow Graph Nodes

CanvasID
Unique identifier of the element's canvas.

ElementID
Unique identifier of the element.

Node Output

StepValue
The smallest increment allowed between values of the element identified by ElementID. Zero means no restriction.

UI:Slider:SetStepValue Node
Sets the smallest increment allowed between values. Zero means no restriction.

Node Inputs

Activate
Sets the smallest increment allowed between values. Zero means no restriction.

CanvasID
Unique identifier of the element's canvas.

ElementID
Unique identifier of the element.

StepValue
The smallest increment allowed between values of the element identified by ElementID. Zero means no restriction.

Node Output

Done
Sends a signal when the node's action is finished.

UI:Slider:GetTrackEntity Node
Gets the track element.

Node Inputs

Activate
Updates the output.

CanvasID
Unique identifier of the element's canvas.

ElementID
Unique identifier of the element.
Node Output

Track
The track element.

UI:Slider:SetTrackEntity Node
Sets the track element.

Node Inputs

Activate
Sets the track element.

CanvasID
Unique identifier of the element's canvas.

ElementID
Unique identifier of the element.

Track
The track element.

Node Output

Done
Sends a signal when the node's action is finished.

UI:Slider:GetValue Node
Gets the value of slider.

Node Inputs

Activate
Updates the output.

CanvasID
Unique identifier of the element's canvas.

ElementID
Unique identifier of the element.

Node Output

Value
The slider value of the element identified by ElementID.

UI:Slider:SetValue Node
Sets the value of the slider.
**Node Inputs**

**Activate**
Sets the value of the slider.

**CanvasID**
Unique identifier of the element's canvas.

**ElementID**
Unique identifier of the element.

**Value**
The slider value of the element identified by **ElementID**.

**Node Output**

**Done**
Sends a signal when the node's action is finished.

**UI:Slider:GetValueChangedActionName Node**
Gets the action triggered when the value is done changing.

**Node Inputs**

**Activate**
Updates the output.

**CanvasID**
Unique identifier of the element's canvas.

**ElementID**
Unique identifier of the element.

**Node Output**

**ValueChangedAction**
The action name.

**UI:Slider:SetValueChangedActionName Node**
Sets the action triggered when the value is done changing.

**Node Inputs**

**Activate**
Updates the output.

**CanvasID**
Unique identifier of the element's canvas.

**ElementID**
Unique identifier of the element.
ElementID

Unique identifier of the element.

ValueChangedAction

The action name.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Slider:GetValueChangingActionName Node

Gets the action triggered while the value is changing.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

ValueChangingAction

The action name.

UI:Slider:SetValueChangingActionName Node

Sets the action triggered while the value is changing.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

ValueChangingAction

The action name.
Node Output

Done

Sends a signal when the node's action is finished.

UI Text Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard’s new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

The UI Text component flow graph nodes have been superseded by the UIe Text component flow graph nodes. For best results, use the UIe Text component flow graph nodes.

Use the following flow graph nodes to perform actions on the text component.

UI:Text:GetColor Node

Gets the color to draw the text string.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Outputs

Color

The RGB value (0 – 255 each for R, G, and B) of the element identified by ElementID.

Alpha

The alpha value (0 – 255) of the element identified by ElementID.

UI:Text:SetColor Node

Sets the color to draw the text string.

Node Inputs

Activate

Updates the output.
CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Color

The RGB value (0 – 255 each for R, G, and B) of the element identified by ElementID.

Alpha

The alpha value (0 – 255) of the element identified by ElementID.

Node Output

Done

Sends a signal when the node's action is finished.

UI:Text:GetFont Node

Gets the path to the font.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

Font

The path to the font used by the element.

UI:Text:SetFontNode

Sets the path to the font.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.
Font

The path to the font used by the element identified by \texttt{ElementID}.

Node Output

Done

Sends a signal when the node's action is finished.

\texttt{UI:Text:GetFontSize Node}

Gets the font size in points.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

FontSize

The font size of the element identified by \texttt{ElementID}.

\texttt{UI:Text:SetFontSize Node}

Sets the font size in points.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

FontSize

The font size of the element identified by \texttt{ElementID}.

Node Output

Done

Sends a signal when the node's action is finished.
UI:Text:GetOverflowMode Node

Gets the overflow behavior of the text.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

OverflowMode

An integer representing how overflow text is handled.

Valid values: 0 = Overflow text | 1 = Clip text

UI:Text:SetOverflowModeNode

Sets the overflow behavior of the text.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

OverflowMode

An integer representing how overflow text is handled.

Valid values: 0 = Overflow text | 1 = Clip text

Node Output

Done

Sends a signal when the node's action is finished.

UI:Text:GetText Node

Gets the text string that the element displays.
**Node Inputs**

**Activate**

Updates the output.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

---

**Node Output**

**Value**

The text string being displayed by the element identified by **`ElementID`**.

---

**UI:Text:SetText Node**

Sets the text string being displayed by the element.

**Node Inputs**

**Activate**

Updates the output.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**Value**

The text string being displayed by the element identified by **`ElementID`**.

---

**Node Output**

**Done**

Sends a signal when the node's action is finished.

---

**UI:Text:GetWrapText Node**

Gets whether text is wrapped.

**Node Inputs**

**Activate**

Updates the output.

**CanvasID**

Unique identifier of the element's canvas.
ElementID

Unique identifier of the element.

Node Output

WrapTextSetting

An integer representing how long text lines are handled.

Valid values: 0 = No wrap | 1 = Wrap

UI:Text:SetWrapText Node

Gets whether text is wrapped.

Node Inputs

Activate

Updates the outputs.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

WrapTextSetting

An integer representing how long text lines are handled.

Valid values: 0 = No wrap | 1 = Wrap

Node Output

Done

Sends a signal when the node's action is finished.

UI Text Input Component Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

The UI Text Input component flow graph nodes have been superseded by the UIe Text Input (p. 547) component flow graph nodes. For best results, use the UIe Text Input (p. 547) component flow graph nodes.

Use the following flow graph nodes to perform actions on the text input component.
UI:TextInput:GetChangeAction Node
Gets the action triggered when the text is changed.

Node Inputs

Activate
Updates the output.

CanvasID
Unique identifier of the element's canvas.

ElementID
Unique identifier of the element.

Node Output

ChangeAction
The action name.

UI:TextInput:SetChangeAction Node
Sets the action triggered when the text is changed.

Node Inputs

Activate
Updates the output.

CanvasID
Unique identifier of the element's canvas.

ElementID
Unique identifier of the element.

ChangeAction
The action name.

Node Output

Done
Sends a signal when the node's action is finished.

UI:TextInput:GetCursorBlinkInterval Node
Gets the cursor blink interval of the text input.

Node Inputs

Activate
Updates the output.
CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

CursorBlinkInterval

The cursor blink in interval of the element identified by ElementID.

UI:TextInput:SetCursorBlinkInterval Node

Gets the cursor blink interval of the text input.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

CursorBlinkInterval

The cursor blink in interval of the element identified by ElementID.

Node Output

Done

Sends a signal when the node's action is finished.

UI:TextInput:GetEndEditAction Node

Gets the action triggered when the editing of text is finished.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.
Node Output

**EndEditAction**

The action name.

**UI:TextInput:SetEndEditAction Node**

Sets the action triggered when the editing of text is finished.

**Node Inputs**

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

**EndEditAction**

The action name.

**Node Output**

**Done**

Sends a signal when the node's action is finished.

**UI:TextInput:GetEnterAction Node**

Gets the action triggered when **Enter** is pressed.

**Node Inputs**

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

**Node Output**

**EnterAction**

The action name.

**UI:TextInput:SetEnterAction Node**

Sets the action triggered when **Enter** is pressed.
Node Inputs

Activate
 Updates the output.

CanvasID
 Unique identifier of the element's canvas.

ElementID
 Unique identifier of the element.

EnterAction
 The action name.

Node Output

Done
 Sends a signal when the node's action is finished.

UI:TextInput:GetIsPasswordField Node
 Gets whether the text input is configured as a password field.

Node Inputs

Activate
 Updates the output.

CanvasID
 Unique identifier of the element's canvas.

ElementID
 Unique identifier of the element.

Node Output

IsPasswordField
 Boolean. Whether the element identified by ElementID is configured as a password field.

UI:TextInput:SetIsPasswordField Node
 Sets whether the text input is configured as a password field.

Node Inputs

Activate
 Updates the output.

CanvasID
 Unique identifier of the element's canvas.
ElementID

Unique identifier of the element.

IsPasswordField

Boolean. Whether the element identified by ElementID is configured as a password field.

Node Output

Done

Sends a signal when the node's action is finished.

UI:TextInput:GetMaxStringLength Node

Gets the maximum number of characters that can be entered.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

Node Output

MaxStringLength

An integer representing the maximum number of characters that can be entered.

Valid values: 0 = none allowed | -1 = unlimited

UI:TextInput:SetMaxStringLength Node

Sets the maximum number of characters that can be entered.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

MaxStringLength

An integer representing the maximum number of characters that can be entered.
Valid values: 0 = none allowed | -1 = unlimited

**Node Output**

**Done**

Sends a signal when the node's action is finished.

**UI:TextInput:GetPlaceHolderTextEntity Node**

Gets the placeholder text element.

**Node Inputs**

**Activate**

Updates the output.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**Node Output**

**PlaceHolderTextElement**

The placeholder text element.

**UI:TextInput:SetPlaceHolderTextEntity Node**

Sets the placeholder text element.

**Node Inputs**

**Activate**

Updates the output.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**PlaceHolderTextElement**

The placeholder text element.

**Node Output**

**Done**

Sends a signal when the node's action is finished.
**UI:TextInput:GetText Node**

Gets the text string that the element is displaying or allowing to be edited.

**Node Inputs**

- **Activate**
  - Updates the output.
- **CanvasID**
  - Unique identifier of the element's canvas.
- **ElementID**
  - Unique identifier of the element.

**Node Output**

- **Value**
  - The text string being displayed or edited by the element

**UI:TextInput:SetText Node**

Sets the text string that the element is displaying or allowing to be edited.

**Node Inputs**

- **Activate**
  - Updates the output.
- **CanvasID**
  - Unique identifier of the element's canvas.
- **ElementID**
  - Unique identifier of the element.
- **Value**
  - The text string being displayed or edited by the element

**Node Output**

- **Done**
  - Sends a signal when the node's action is finished.

**UI:TextInput:GetTextCursorColor Node**

Gets the color to be used for the text cursor.

**Node Inputs**

- **Activate**
  - Updates the output.
CanvasID
Unique identifier of the element's canvas.

ElementID
Unique identifier of the element.

**Node Outputs**

**Color**
The RGB value (0 – 255 each for R, G, and B) of the element identified by `ElementID`.

**Alpha**
The alpha value (0 – 255) of the element identified by `ElementID`.

**UI:TextInput:SetTextCursorColor Node**
Sets the color to be used for the text cursor.

**Node Inputs**

**Activate**
Updates the output.

CanvasID
Unique identifier of the element's canvas.

ElementID
Unique identifier of the element.

Color
The RGB value (0 – 255 each for R, G, and B) of the element identified by `ElementID`.

Alpha
The alpha value (0 – 255) of the element identified by `ElementID`.

**Node Output**

**Done**
Sends a signal when the node's action is finished.

**UI:TextInput:GetTextEntity Node**
Gets the text element.

**Node Inputs**

**Activate**
Updates the output.

CanvasID
Unique identifier of the element's canvas.
ElementID

Unique identifier of the element.

Node Output

TextElement

The text element.

UI:TextInput:SetTextEntity Node

Gets the text element.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

TextElement

The text element.

Node Output

Done

Sends a signal when the node's action is finished.

UI:TextInput:GetTextSelectionColor Node

Gets the color to be used for the text background when it is selected.

Node Inputs

Activate

Updates the output.

CanvasID

Unique identifier of the element's canvas.

ElementID

Unique identifier of the element.

TextElement

The text element.

Node Outputs

Color

The RGB value (0 – 255 each for R, G, and B) of the element identified by ElementID.
**Alpha**

The alpha value (0 – 255) of the element identified by `ElementID`.

**UI:TextInput:SetTextSelectionColor Node**

 Gets the color to be used for the text background when it is selected.

**Node Inputs**

**Activate**

Updates the output.

**CanvasID**

Unique identifier of the element's canvas.

**ElementID**

Unique identifier of the element.

**Color**

The RGB value (0 – 255 for R, G, and B) of the element identified by `ElementID`.

**Alpha**

The alpha value (0 – 255) of the element identified by `ElementID`.

**Node Output**

**Done**

Sends a signal when the node's action is finished.

**UI Animation Node**

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 about the latest features, see the Amazon Lumberyard User Guide.

The UI animation flow graph node has been superseded by the UIe Animation (p. 559) flow graph node. For best results, use the UIe Animation (p. 559) flow graph node.

The UI animation node consists of the following node inputs and outputs:

**UI:Sequence:Play Node**

Controls playback of a UI animation sequence.

**Node Inputs**

**Start**

Starts playing the sequence from the beginning and triggers the `OnStarted` output.

**Stop**

Jumps the animation to the end and stops playing and triggers the `OnStopped` output.
Abort
Jumps the animation to the end and stops playing and triggers the OnAborted output.

Pause
Pauses the animation.

Resume
Continues playing a previously paused animation.

Reset
Resets the animation to the start. This applies all the key values for the first keyframe of the animation.

CanvasID
Unique ID of UI canvas that contains the animation sequence.

SequenceName
The name of the sequence to play.

Node Outputs

OnStarted
Triggers when the sequence starts playing.

OnStopped
Triggers an output when the sequence stops playing, either because the end of the animation is reached or because the sequence is forced to stop (for example, by using the Stop node input).

OnAborted
Triggers an output when the sequence is aborted (for example, by using the Abort node input).

Associating Canvases with UI Flow Graph Nodes

You must associate all UI flow graph nodes with a UI canvas. There are two sets of flow graph nodes for the UI: Ule and UI. The Ule set of flow graph nodes supersedes the now-legacy UI set of flow graph nodes.

In the Ule set of nodes, you assign a special entity to the node’s Choose Entity input using either the new Component Entity system or the legacy Entity system. These procedures are described in this section.

In the legacy UI set of nodes, the CanvasID comes from the UI:Canvas:Load node.

Using the Component Entity system to associate a UI canvas with a Ule flow graph node

1. In the viewport, right-click and select Create Component Entity.
2. Right-click the newly created component and click Flow Graph, Add.

   Enter a name for the flow graph, or leave it as Default.
3. If the Flow Graph Editor is not yet open, click Tools, Flow Graph.
4. In the viewport, right-click the component entity and select Flow Graph, Open, <flow graph name>.
5. Select the newly created component entity. In the Entity Inspector, add a UI Canvas Asset Ref component and enter a path to the canvas you want to associate.
6. In the Flow Graph Editor, in the flow graph you created, add any UIe flow graph node to the graph.
7. Right-click the node you placed and do one of the following:
   • Click Assign graph entity if the canvas you want to reference is selected in the UI Canvas Asset Ref component.
   • Select a different entity and then click Assign selected entity to reference a different canvas.

   Note
   This other entity can be either a component entity with the UI Canvas Asset Ref component on it or a legacy entity that is a UiCanvasRefEntity.

Using the Legacy Entity system to associate a UI canvas with a UIe flow graph node

1. In Lumberyard Editor's Rollup Bar, on the Objects tab, click Entity. Expand the UI folder and drag UiCanvasRefEntity into the viewport.
2. Select the newly created UiCanvasRefEntity entity. In its Entity Properties, click CanvasPath and enter a path to the canvas you want to associate.
3. In Lumberyard Editor, click Tools, Flow Graph.
4. In the Flow Graph editor's Graphs pane, select a flow graph.
5. Add any UIe flow graph node to the graph.
6. Right-click the node you placed and do one of the following:
   • Click Assign graph entity if the flow graph is associated with the UiCanvasRefEntity.
   • If the flow graph you used is not associated with the UiCanvasRefEntity, make sure the UiCanvasRefEntity is selected in your viewport and then click Assign selected entity.
To associate a UI canvas with a legacy UI flow graph node

1. Load a canvas in the Flow Graph editor. See Loading Canvases in the Flow Graph Editor (p. 644) for more information.
2. Add any UI flow graph node to the graph.
3. Connect the CanvasID output of the UI:Canvas:Load node to the CanvasID input of the new node.

```
UI:Canvas:Load
  ➔ Activate
  ➔ CanvasPathname=UI/Canvases/test.uicanvas ➔ CanvasID
  ➔ Disabled=0
```

Loading Canvases in the Flow Graph 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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the Flow Graph Editor to load and unload UI canvases.

For more information about using flow graphs, see Flow Graph System (p. 218).

For more information about the flow graph nodes you can use to make elements and components respond to user input, see UI Flow Graph Nodes (p. 482).

You can load canvases in the Flow Graph Editor using either the Ui node set (recommended) or the legacy UI node set. You can also load a canvas automatically using a component entity (without using flow graph). These procedures are described in this section.

```
Game:Start
  ➔ InGame=1 ➔ output
  ➔ InEditor=1
```

```
UIe:Canvas:LoadIntoEntity
  ➔ <Graph Entity>
  ➔ Activate ➔ OnLoad
  ➔ Disabled=0
```

Use the following procedure to load canvases using the Ui node set. This is the recommended method of loading canvases in flow graph.
To load a canvas in the Flow Graph Editor using the Ule node set

1. In Lumberyard Editor, click **Tools, Flow Graph**.
2. In the **Flow Graph** editor, select a flow graph from the **Graphs** pane.
3. Right-click anywhere in the graphs pane and select **Add Node, Game, Start**.
4. Right-click anywhere in the graphs pane and select **Add Node, Ule, Canvas, LoadIntoEntity**.
5. Right-click the **Ule:Canvas:LoadIntoEntity** node and select **Assign selected entity** or **Assign graph entity** to assign a UI canvas reference entity to the node.

   For more information about assigning a UI canvas reference entity to the node, see **Associating Canvases with UI Flow Graph Nodes** (p. 642).
6. Connect the **Game:Start** node output to the **Activate** input on the **Ule:Canvas:LoadIntoEntity** node.

To use the legacy UI node set to load canvases, use the following procedure.

**To load a canvas in the Flow Graph Editor using the legacy UI node set**

1. In Lumberyard Editor, click **Tools, Flow Graph**.
2. In the **Flow Graph** editor, select a flow graph from the **Graphs** pane.
3. Right-click anywhere in the graphs pane and select **Add Node, Game, Start**.
4. Right-click anywhere in the graphs pane and select **Add Node, UI, Canvas, Load**.
5. Connect the **Game:Start** node output to the **Activate** input on the **UI:Canvas:Load** node.
6. Double-click **CanvasPathname** in the **UI:Canvas:Load** node, and type a path in the **CanvasPathname** text box or use the file browser to navigate to the path. The path is relative to the project folder.

   **Note**
   You might need to zoom in to be able to edit **CanvasPathname**.

The following method uses the component entity system to load a canvas, without using any flow graphs.

**Using the Component Entity system to load a UI canvas**

1. In the level, create a component entity.
2. In the **Entity Inspector**, add to this component entity a **UI Canvas Asset Ref** to specify the UI canvas and optionally to automatically load it when the level loads.
3. Select the **Load automatically** check box.
Vec3 Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to define vector (Vec3) operations.

Topics

- AddVec3 node (p. 647)
- Calculate node (p. 647)
- ClampVec3 node (p. 648)
- CrossVec3 node (p. 648)
- DotVec3 node (p. 649)
- EqualVec3 node (p. 649)
- FromVec3 node (p. 649)
• MagnitudeVec3 node (p. 650)
• MulVec3 node (p. 650)
• NormalizeVec3 node (p. 651)
• ReciprocalVec3 node (p. 651)
• RotateVec3onAxis node (p. 651)
• ScaleVec3 node (p. 652)
• SetVec3 node (p. 652)
• SubVec3 node (p. 653)
• ToVec3 node (p. 653)

AddVec3 node

Used to output the sum of two vectors.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>A</td>
<td>Vec3</td>
<td>First operand</td>
</tr>
<tr>
<td>B</td>
<td>Vec3</td>
<td>Second operand</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Vec3</td>
<td>Addition of A and B</td>
</tr>
</tbody>
</table>

Calculate node

Used to output the specified calculation between two vectors.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>Operator</td>
<td>Integer</td>
<td>Math operation to perform</td>
</tr>
<tr>
<td>A</td>
<td>Vec3</td>
<td>First operand</td>
</tr>
<tr>
<td>B</td>
<td>Vec3</td>
<td>Second operand</td>
</tr>
</tbody>
</table>
**ClampVec3 node**

Used to clamp the output range of a vector between a minimum and a maximum.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Vec3</td>
<td>Input value</td>
</tr>
<tr>
<td>Min</td>
<td>Vec3</td>
<td>Minimum clamping value</td>
</tr>
<tr>
<td>Max</td>
<td>Vec3</td>
<td>Maximum clamping value</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Vec3</td>
<td>Triggers when the input value is between the minimum and maximum values</td>
</tr>
</tbody>
</table>

**CrossVec3 node**

Used to output the cross product of two vectors.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>A</td>
<td>Vec3</td>
<td>First operand</td>
</tr>
<tr>
<td>B</td>
<td>Vec3</td>
<td>Second operand</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Vec3</td>
<td>Outputs the cross product of the inputs</td>
</tr>
</tbody>
</table>
**DotVec3 node**

Used to output the dot product of the inputs.

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>A</td>
<td>Vec3</td>
<td>First operand</td>
</tr>
<tr>
<td>B</td>
<td>Vec3</td>
<td>Second operand</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Float</td>
<td>Outputs the dot product of the inputs</td>
</tr>
</tbody>
</table>

**EqualVec3 node**

Used to trigger an output when both vectors are equal in value.

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Vec3</td>
<td>First operand</td>
</tr>
<tr>
<td>B</td>
<td>Vec3</td>
<td>Second operand</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Boolean</td>
<td>Triggers when A and B are equal in value</td>
</tr>
</tbody>
</table>

**FromVec3 node**

Used to output the x, y, and z values of the vector.
### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vec3</td>
<td>Vec3</td>
<td>Input vector</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Float</td>
<td>X-axis value of vector</td>
</tr>
<tr>
<td>Y</td>
<td>Float</td>
<td>Y-axis value of vector</td>
</tr>
<tr>
<td>Z</td>
<td>Float</td>
<td>Z-axis value of vector</td>
</tr>
</tbody>
</table>

#### MagnitudeVec3 node

Used to output the magnitude (length) of the vector.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td>Vec3</td>
<td>Input vector</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>Any</td>
<td>Magnitude (length) of the input vector</td>
</tr>
</tbody>
</table>

#### MulVec3 node

Used to output the multiplication of two vectors.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>A</td>
<td>Vec3</td>
<td>First operand</td>
</tr>
<tr>
<td>B</td>
<td>Vec3</td>
<td>Second operand</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Vec3</td>
<td>Multiplication of A and B</td>
</tr>
</tbody>
</table>

**NormalizeVec3 node**

Used to output the normalized value of the vector.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td>Vec3</td>
<td>Vector input</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Vec3</td>
<td>Normalized vector input</td>
</tr>
<tr>
<td>Length</td>
<td>Float</td>
<td>Magnitude</td>
</tr>
</tbody>
</table>

**ReciprocalVec3 node**

Used to output the reciprocal of the vector.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td>Vec3</td>
<td>Input vector</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>Float</td>
<td>Reciprocal value of input</td>
</tr>
</tbody>
</table>

**RotateVec3onAxis node**

Used to output an axis-rotated value of the vector.
# Lumberyard Legacy Reference

## Vec3 Nodes

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>Vector</td>
<td>Vec3</td>
<td>Input vector to rotate</td>
</tr>
<tr>
<td>Axis</td>
<td>Vec3</td>
<td>Axis to rotate input around</td>
</tr>
<tr>
<td>Angle</td>
<td>Float</td>
<td>Angle in degrees to rotate</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rotated Vector</td>
<td>Vec3</td>
<td>Result of the rotation</td>
</tr>
</tbody>
</table>

**ScaleVec3 node**

Used to output a scaled value of the vector.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td>Vec3</td>
<td>Input vector</td>
</tr>
<tr>
<td>Scale</td>
<td>Float</td>
<td>Scale factor to apply to the input</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Vec3</td>
<td>Result of the scaling</td>
</tr>
</tbody>
</table>

**SetVec3 node**

Used to output the input value when the Set input is activated.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>Any</td>
<td>Triggers the vector to the output</td>
</tr>
<tr>
<td>In</td>
<td>Vec3</td>
<td>Input value</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Vec3</td>
<td>Outputs the input value</td>
</tr>
</tbody>
</table>

**SubVec3 node**

Used to output the subtracted value of two vectors.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Triggers the node</td>
</tr>
<tr>
<td>A</td>
<td>Vec3</td>
<td>First operand</td>
</tr>
<tr>
<td>B</td>
<td>Vec3</td>
<td>Second operand</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out</td>
<td>Vec3</td>
<td>Subtraction of B from A</td>
</tr>
</tbody>
</table>

**ToVec3 node**

Used to output three floating point values to a vector.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Float</td>
<td>X-axis value</td>
</tr>
<tr>
<td>Y</td>
<td>Float</td>
<td>Y-axis value</td>
</tr>
<tr>
<td>Z</td>
<td>Float</td>
<td>Z-axis value</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
<td>Vec3</td>
<td>Vector output</td>
</tr>
</tbody>
</table>
Vehicle Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard’s new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to configure vehicle behavior and related settings.

**Note**
These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at Lumberyard Downloads.

**Topics**
- Attachment node (p. 654)
- ChangeSeat node (p. 655)
- ChaseTarget node (p. 655)
- Damage node (p. 656)
- Destroy node (p. 657)
- Enter node (p. 657)
- FollowPath node (p. 658)
- GetSeatHelper node (p. 658)
- Handbrake node (p. 659)
- Honk node (p. 659)
- Lights node (p. 659)
- Lock node (p. 660)
- MoveActionMult node (p. 660)
- Movement node (p. 660)
- MovementParams node (p. 661)
- Passenger node (p. 661)
- Seat node (p. 662)
- StickPath node (p. 662)
- Turret node (p. 663)
- Unload node (p. 663)

**Attachment node**

Used to control vehicle entity attachments.
Vehicle Nodes

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attachment</td>
<td>String</td>
<td>Attachment to add</td>
</tr>
<tr>
<td>EntityId</td>
<td>Any</td>
<td>ID of the entity to use</td>
</tr>
<tr>
<td>Attach</td>
<td>Any</td>
<td>Attaches the item</td>
</tr>
<tr>
<td>Detach</td>
<td>Any</td>
<td>Detaches the item</td>
</tr>
</tbody>
</table>

ChangeSeat node

Used to move a character from one seat to another one. Only works if the character is already inside a vehicle.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>Any</td>
<td>Triggers the seat change</td>
</tr>
<tr>
<td>Seat</td>
<td>Integer</td>
<td>Seat to change to</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Succeed</td>
<td>Any</td>
<td>Seat change succeeded</td>
</tr>
<tr>
<td>Fail</td>
<td>Any</td>
<td>Seat change failed</td>
</tr>
</tbody>
</table>

ChaseTarget node

Used to follow or navigate along the specified path while attempting to establish line of sight or fire with the specified target.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>Any</td>
<td>Triggers the chase</td>
</tr>
</tbody>
</table>
### Vehicle Nodes

**Port** | **Type** | **Description**
---|---|---
Cancel | Any | Cancels the chase
Path Name | String | Name of the path to follow
Max speed of the vehicle | Float | Maximum speed of the vehicle
Min Distance | Float | Minimum chase distance to the target
Max Distance | Float | Minimum chase distance to the target
Target | Any | ID of the target to chase
Force | Integer | Force execution method

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail</td>
<td>Any</td>
<td>Chase failed</td>
</tr>
</tbody>
</table>

### Damage node

Used to handle vehicle damage.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HitTrigger</td>
<td>Any</td>
<td>Triggers that causes the vehicle to sustain damage</td>
</tr>
<tr>
<td>HitValue</td>
<td>Float</td>
<td>Amount of damage the vehicle will sustain</td>
</tr>
<tr>
<td>HitPosition</td>
<td>Vec3</td>
<td>Position at which the vehicle will sustain the hit</td>
</tr>
<tr>
<td>HitRadius</td>
<td>Float</td>
<td>Radius of the hit</td>
</tr>
<tr>
<td>Indestructible</td>
<td>Boolean</td>
<td>Value of true sets the vehicle to be indestructible</td>
</tr>
<tr>
<td>HitType</td>
<td>String</td>
<td>Type of damage</td>
</tr>
<tr>
<td>HitComponent</td>
<td>String</td>
<td>Vehicle component that will receive the hit</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Damaged</td>
<td>Float</td>
<td>Amount of damage sustained by the vehicle</td>
</tr>
</tbody>
</table>
### Vehicle Nodes

#### Destroy node

Used to destroy the vehicle.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destroy</td>
<td>Any</td>
<td>Trigger to destroy the vehicle</td>
</tr>
</tbody>
</table>

#### Enter node

Used to make an AI agent sit in a specified seat of a specified vehicle.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Cancel</td>
<td>Any</td>
<td>Cancels the operation</td>
</tr>
<tr>
<td>VehicleId</td>
<td>Any</td>
<td>ID of the vehicle</td>
</tr>
<tr>
<td>Seat</td>
<td>Integer</td>
<td>Seat to sit on</td>
</tr>
<tr>
<td>Fast</td>
<td>Boolean</td>
<td>Skip approach and enter vehicle</td>
</tr>
<tr>
<td>Force</td>
<td>Integer</td>
<td>Force execution method</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Action completed</td>
</tr>
<tr>
<td>Succeed</td>
<td>Any</td>
<td>Action was successful</td>
</tr>
<tr>
<td>Fail</td>
<td>Any</td>
<td>Action failed</td>
</tr>
</tbody>
</table>
FollowPath node

Used to follow the path speed stance action.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Cancel</td>
<td>Any</td>
<td>Cancels execution</td>
</tr>
<tr>
<td>PathFindToStart</td>
<td>Boolean</td>
<td>Whether to find the start of the path</td>
</tr>
<tr>
<td>Reverse</td>
<td>Boolean</td>
<td>Reverses the path direction</td>
</tr>
<tr>
<td>StartNearest</td>
<td>Boolean</td>
<td>Starts the path at the nearest point on path</td>
</tr>
<tr>
<td>Loops</td>
<td>Integer</td>
<td>Number of times to loop around the path</td>
</tr>
<tr>
<td>Path Name</td>
<td>String</td>
<td>Name of the path</td>
</tr>
<tr>
<td>Speed (m/s)</td>
<td>Float</td>
<td>Speed in meters/second</td>
</tr>
<tr>
<td>Force</td>
<td>Integer</td>
<td>Force execution method</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Action completed</td>
</tr>
<tr>
<td>Succeed</td>
<td>Any</td>
<td>Action was successful</td>
</tr>
<tr>
<td>Fail</td>
<td>Any</td>
<td>Action failed</td>
</tr>
</tbody>
</table>

GetSeatHelper node

Used to get the helper position of a seat for entering the vehicle.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>Any</td>
<td>Get helper position</td>
</tr>
<tr>
<td>Seat</td>
<td>Integer</td>
<td>Seat to be entered</td>
</tr>
</tbody>
</table>
## Vehicle Nodes

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>Vec3</td>
<td>Position of seat helper</td>
</tr>
<tr>
<td>Dir</td>
<td>Vec3</td>
<td>Direction of seat helper</td>
</tr>
</tbody>
</table>

### Handbrake node

Used to toggle the vehicle handbrake. Currently only supported for the ArcadeWheeled movement type.

![Handbrake node](handbrake_node.png)

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates the vehicle handbrake</td>
</tr>
<tr>
<td>Deactivate</td>
<td>Any</td>
<td>Deactivates the vehicle handbrake</td>
</tr>
<tr>
<td>ResetTimer</td>
<td>Float</td>
<td>Resets the timer</td>
</tr>
</tbody>
</table>

### Honk node

Use to control a vehicle's horn.

![Honk node](honk_node.png)

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Activates the vehicle horn</td>
</tr>
<tr>
<td>Duration</td>
<td>Float</td>
<td>Duration in seconds of the horn</td>
</tr>
</tbody>
</table>

### Lights node

Used to control a vehicle's lights.

![Lights node](lights_node.png)

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LightType</td>
<td>String</td>
<td>Type of vehicle light</td>
</tr>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Activates vehicle lights</td>
</tr>
</tbody>
</table>
## Lumberyard Legacy Reference

### Vehicle Nodes

#### Deactivate

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deactivate</td>
<td>Any</td>
<td>Deactivates vehicle lights</td>
</tr>
</tbody>
</table>

#### Lock node

*Used to lock or unlock all seats of a vehicle.*

- **Inputs**
  - Port: Lock
    - Type: Any
    - Description: Locks the vehicle
  - Port: Unlock
    - Type: Any
    - Description: Unlocks the vehicle
  - Port: LockType
    - Type: Integer
    - Description: Type of vehicle lock

#### MoveActionMult node

*Used to add multipliers to a vehicle's movement actions.*

- **Inputs**
  - Port: EnableTrigger
    - Type: Any
    - Description: Activates the node
  - Port: DisableTrigger
    - Type: Any
    - Description: Deactivates the node
  - Port: PowerMult
    - Type: Float
    - Description: Vehicle engine power multiplier
  - Port: RotatePitch
    - Type: Float
    - Description: Vehicle pitch rotation multiplier
  - Port: RotateYaw
    - Type: Float
    - Description: Vehicle yaw rotation multiplier
  - Port: PowerMustBePositive
    - Type: Boolean
    - Description: True if power multiplication is positive (increase)

#### Movement node

*Used to control vehicle movement.*

- **Inputs**
  - Port: WarmupEngineTrigger
    - Type: Integer
    - Description: Vehicle engine warmup trigger
  - Port: ZeroMass
    - Type: Boolean
    - Description: True if vehicle mass is zero
  - Port: RestoreMass
    - Type: Boolean
    - Description: True if vehicle mass is restored
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WarmUpEngineTrigger</td>
<td>Boolean</td>
<td>Warms up vehicle engine</td>
</tr>
<tr>
<td>ZeroMass</td>
<td>Any</td>
<td>Vehicle has zero mass</td>
</tr>
<tr>
<td>RestoreMass</td>
<td>Any</td>
<td>Restores vehicle mass</td>
</tr>
</tbody>
</table>

**MovementParams node**

Used to modify vehicle movement parameters.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>MaxSpeedFactor</td>
<td>Float</td>
<td>Maximum vehicle speed factor</td>
</tr>
<tr>
<td>AccelerationFactor</td>
<td>Float</td>
<td>Maximum vehicle acceleration factor</td>
</tr>
</tbody>
</table>

**Passenger node**

Used to manage vehicle passengers.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActorInTrigger</td>
<td>Any</td>
<td>Forces actor to get into vehicle if a seat is available</td>
</tr>
<tr>
<td>ActorOutTrigger</td>
<td>Any</td>
<td>Forces actor to get out of the vehicle</td>
</tr>
<tr>
<td>ActorId</td>
<td>Any</td>
<td>ID of the action</td>
</tr>
<tr>
<td>SeatId</td>
<td>Integer</td>
<td>ID of the seat</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActorIn</td>
<td>Any</td>
<td>Triggered if any actor got into vehicle</td>
</tr>
<tr>
<td>ActorOut</td>
<td>Any</td>
<td>Triggered if any actor got out of vehicle</td>
</tr>
</tbody>
</table>
**Seat node**

Used to manage vehicle seats.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seat</td>
<td>Integer</td>
<td>ID of seat</td>
</tr>
<tr>
<td>SeatName</td>
<td>String</td>
<td>Name of seat</td>
</tr>
<tr>
<td>IsDriverSeat</td>
<td>Boolean</td>
<td>True is driver seat</td>
</tr>
<tr>
<td>Lock</td>
<td>Any</td>
<td>Locks the vehicle</td>
</tr>
<tr>
<td>Unlock</td>
<td>Any</td>
<td>Unlocks the vehicle</td>
</tr>
<tr>
<td>LockType</td>
<td>Integer</td>
<td>Type of vehicle lock</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeatId</td>
<td>Integer</td>
<td>ID of seat</td>
</tr>
<tr>
<td>PassengerId</td>
<td>Integer</td>
<td>ID of passenger</td>
</tr>
</tbody>
</table>

**StickPath node**

Used to follow the specified path to the end and sticking to the optional target, either continuously or as a one-off event.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Cancel</td>
<td>Any</td>
<td>Cancels execution</td>
</tr>
<tr>
<td>Path Name</td>
<td>String</td>
<td>Name of path</td>
</tr>
</tbody>
</table>
### Vehicle Nodes

#### Continuous
- **Type:** Boolean
- **Description:** Whether vehicle can continue to follow the path or stops once it reaches the target

#### CanReverse
- **Type:** Boolean
- **Description:** Whether vehicle is allowed to drive in reverse to follow target or path

#### Max speed of the vehicle
- **Type:** Float
- **Description:** Maximum speed of the vehicle

#### Min Distance
- **Type:** Float
- **Description:** Minimum stick distance to the target

#### Max Distance
- **Type:** Float
- **Description:** Maximum stick distance to the target

#### Target
- **Type:** Any
- **Description:** ID of target to stick to when following the path

#### Force
- **Type:** Integer
- **Description:** Force execution method

### Outputs

#### Done
- **Type:** Any
- **Description:** Action completed

#### Succeed
- **Type:** Any
- **Description:** Action was successful

#### Fail
- **Type:** Any
- **Description:** Action failed

#### Close
- **Type:** Any
- **Description:** Close to destination

### Turret node

Use to control the vehicle turret.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>Any</td>
<td>Activates vehicle turret</td>
</tr>
<tr>
<td>SeatId</td>
<td>Integer</td>
<td>ID of seat</td>
</tr>
<tr>
<td>AimAngles</td>
<td>Vec3</td>
<td>Turret aiming angle</td>
</tr>
<tr>
<td>AimPos</td>
<td>Vec3</td>
<td>Turret target location</td>
</tr>
</tbody>
</table>

### Unload node

Use to unload a vehicle, ejecting specified passengers.
Lumberyard Legacy Reference
Video Nodes

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync</td>
<td>Any</td>
<td>Triggers the action</td>
</tr>
<tr>
<td>Cancel</td>
<td>Any</td>
<td>Cancels execution</td>
</tr>
<tr>
<td>Seat</td>
<td>Integer</td>
<td>Seat to eject passenger from</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Any</td>
<td>Action completed</td>
</tr>
<tr>
<td>Succeed</td>
<td>Any</td>
<td>Action succeeded</td>
</tr>
<tr>
<td>Fail</td>
<td>Any</td>
<td>Action failed</td>
</tr>
</tbody>
</table>

Video Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to configure video settings.

Topics
- ClipCapture node (p. 664)

ClipCapture node

Used to capture video clips while a game is running and save them locally or to the cloud.
## VideoPlayback Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard’s new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to configure video playback settings.

### Topics
- Play Node (p. 665)
- Pause Nodes (p. 666)
- Stop Nodes (p. 666)
- IsPlaying Nodes (p. 666)
- PlaybackEvents Nodes (p. 667)

### Play Node

Plays the video on the defined entity with the VideoPlayback component.
**VideoPlayback Nodes**

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Boolean</td>
<td>Value of true begins playing video.</td>
</tr>
<tr>
<td>Loop</td>
<td>Boolean</td>
<td>Value of true loops video.</td>
</tr>
<tr>
<td>PlaybackSpeed</td>
<td>Float</td>
<td>Sets speed of playback. Value of 1 is normal speed, 0.5 is half speed, 2.0 is double speed, and so on.</td>
</tr>
</tbody>
</table>

**Pause Nodes**

Pauses the video on the defined entity with the VideoPlayback component.

**Input**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Boolean</td>
<td>Value of true pauses the video.</td>
</tr>
</tbody>
</table>

**Stop Nodes**

Stops the video on the defined entity with the VideoPlayback component.

**Input**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Boolean</td>
<td>Value of true stops the video. If video playback is resumed, it begins at the first frame of the video.</td>
</tr>
</tbody>
</table>

**IsPlaying Nodes**

Determines whether the VideoPlayback component on the defined entity is currently playing video.
# VideoPlayback Nodes

## Input

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Boolean</td>
<td>Value of true triggers output to the Playing output node. Output is triggered whether video is playing or not.</td>
</tr>
</tbody>
</table>

## Output

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Playing</td>
<td>Boolean</td>
<td>Returns true if video is playing, false if video is paused or stopped.</td>
</tr>
</tbody>
</table>

## PlaybackEvents Nodes

Provides a series of outputs that are triggered when the defined entity's VideoPlayback component reaches certain conditions.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Boolean</td>
<td>Value of true triggers node to listen for video playback events on the defined entity.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PlaybackStarted</td>
<td>Boolean</td>
<td>Triggered when the video begins playback.</td>
</tr>
<tr>
<td>PlaybackPaused</td>
<td>Boolean</td>
<td>Triggered when video playback is paused.</td>
</tr>
<tr>
<td>PlaybackStopped</td>
<td>Boolean</td>
<td>Triggered when video playback is stopped.</td>
</tr>
<tr>
<td>PlaybackFinished</td>
<td>Boolean</td>
<td>Triggered when video playback finishes. If a video is set to loop, this is called every time that video finishes a loop.</td>
</tr>
</tbody>
</table>

If a video is not set to loop and finishes without user intervention, this is not triggered; this is triggered only when Stop is directly called.
Virtual Reality (VR) Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use flow graph modules to set up or script your virtual reality game.

The following flow graph modules are available for any attached head-mounted display.

**VR:ControllerTracking**

Provides up-to-date information about any attached motion controller's transform (position and rotation) information in game world space. If an entity is specified in the node, all positions and rotations are specified relative to the entity.

**Node Inputs**

- **Enabled**
  
  Enables or disables the node.

- **Scale**

  Scales the controller's movements.

**Node Outputs**

- **Left pos**
  
  Position of the left controller.

- **Left Rot (PRY)**

  Rotation of the left controller in degrees (PRY – pitch, roll, yaw).

- **Left data ok**

  Valid data output from left controller. This means that the controller is connected and active.

- **Right pos**

  Position of the right controller.

- **Right Rot (PRY)**

  Rotation of the right controller in degrees (PRY – pitch, roll, yaw).

- **Right data ok**

  Valid data output from right controller. This means that the controller is connected and active.

**VR:DeviceInfo**

Gets information about the currently connected device.
Node Input

Activate

Updates the output.

Node Outputs

Name

The name of the active HMD.

RenderWidth

The render width for a single eye (in pixels).

RenderHeight

The render height for a single eye (in pixels).

VerticalFOV

The vertical field of view (FOV) for the HMD in degrees.

HorizontalFOV

The combined horizontal field of view (FOV) for both eyes in degrees.

VR:Dynamics:Controllers

Gives up-to-date information about the current HMD transform (position and rotation) in the game world space.

Node Input

Activate

Updates the outputs.

Node Outputs

Left Controller Active

Boolean. Whether left controller is active and being tracked.

Left Linear Velocity

Vector. Linear velocity of the left controller in local space.

Left Linear Acceleration

Vector. Linear acceleration of the left controller in local space.

Left Angular Velocity

Vector. Angular velocity of the left controller in local space.

Left Angular Acceleration

Vector. Angular acceleration of the left controller in local space.
Right Controller Active

Boolean. Whether right controller is active and being tracked.

Right Linear Velocity

Vector. Linear velocity of the right controller in local space.

Right Linear Acceleration

Vector. Linear acceleration of the right controller in local space.

Right Angular Velocity

Vector. Angular velocity of the right controller in local space.

Right Angular Acceleration

Vector. Angular acceleration of the right controller in local space.

VR:Dynamics:HMD

Provides information about the current angular and linear dynamics of the HMD.

Node Input

Enabled

Enables the node.

Node Outputs

Linear Velocity

Linear velocity of the HMD in local space.

Linear Acceleration

Linear acceleration of the HMD in local space.

Angular Velocity

Angular velocity of the HMD in local space.

Angular Acceleration

Angular acceleration of the HMD in local space.

VR:OpenVR:Playspace

Provides information about the HMD's playspace.

Note

This is now a legacy flow graph node, and has been replaced by VR:Playspace (p. 671).

Node Input

Activate

Updates the outputs.
Node Outputs

Corner0
- The world-space position of corner 0.

Corner1
- The world-space position of corner 1.

Corner2
- The world-space position of corner 2.

Corner3
- The world-space position of corner 3.

Center
- The world-space center of the playspace. Note that the center is on the floor.

Dimensions
- The width (x) and height (y) of the playspace in meters.

IsValid
- If true, the playspace data is valid and configured correctly.

VR:Playspace
Configures the playspace for the Oculus and OpenVR SDKs. The corners represent a rectangle, and run counter-clockwise around the z-axis.

Node Inputs

Activate
- Gets information about the HMD's playspace.

Node Outputs

Corner0
- Vector value. The world-space position of corner 0.

Corner1
- Vector value. The world-space position of corner 1.

Corner2
- Vector value. The world-space position of corner 2.

Corner3
- Vector value. The world-space position of corner 3.

Center
- Vector value. The world-space center of the playspace. Note that the center is on the floor.

Dimensions
- The width and height of the playspace in meters.
IsValid

If true, the playspace data is valid and configured correctly.

VR:RecenterPose

Recenters the view coordinate system for the attached HMD to the current view.

VR:SetTrackingLevel

Sets the TrackingLevel of the attached VR device to 0 (head), 1 (floor), or 2 (fixed). When setting the tracking level to 2, the HMD does not apply translation based on the position of the device, only rotation is applied. This is to aid VR experiences that required a fixed head position, such as viewing 360 degree video.

These options determine how the HMD's origin is calculated for every frame.

VR:TransformInfo

Provides information about the orientation and position of the camera and the HMD.

Node Input

Enabled

Enables the node.

Node Outputs

Camera pos

The position of the current camera in world coordinates.

Camera rot (PRY)

Vector. The orientation of the current camera in world coordinates in degrees (PRY – pitch, roll, yaw).

HMD pos

The position of the HMD with respect to the recentered pose of the tracker.

HMD rot (PRY)

The orientation of the HMD in world coordinates in degrees (PRY – pitch, roll, yaw).

VR:VREnabled

Queries whether VR output is enabled and active in the system. A true output from this node means that an HMD is connected, properly initialized, and being rendered to.

Weapon Nodes

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.
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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to configure weapon settings.

**Note**
These nodes will only work with the Legacy Game Sample (CryEngine GameSDK), which is available at Lumberyard Downloads.

**Topics**
- AmmoChange node (p. 673)
- AutoSightWeapon node (p. 673)
- ChangeFireMode node (p. 674)
- FireWeapon node (p. 674)
- Listener node (p. 675)

### AmmoChange node

Used to give or take ammunition to or from the player. Weapon and ammo type must match.

![AmmoChange node diagram]

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>Any</td>
<td>Activates the node</td>
</tr>
<tr>
<td>Get</td>
<td>Any</td>
<td>Retrieves the amount of ammunition left</td>
</tr>
<tr>
<td>AmmoType</td>
<td>String</td>
<td>Type of ammunition to add</td>
</tr>
<tr>
<td>AmountCount</td>
<td>Integer</td>
<td>Gets the amount of ammunition left</td>
</tr>
<tr>
<td>Add</td>
<td>Boolean</td>
<td>Adds the specified amount of ammunition</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MagazineAmmo</td>
<td>Integer</td>
<td>Ammunition left in the weapon magazine</td>
</tr>
<tr>
<td>InventoryAmmo</td>
<td>Integer</td>
<td>Ammunition left in inventory</td>
</tr>
<tr>
<td>TotalAmmo</td>
<td>Integer</td>
<td>Total ammunition available</td>
</tr>
</tbody>
</table>

### AutoSightWeapon node

This node
Lumberyard Legacy Reference
Weapon Nodes

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enemy</td>
<td>Vec3</td>
<td>Aims the weapon at the enemy's position</td>
</tr>
</tbody>
</table>

ChangeFireMode node

Used to change the weapon fire mode.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch</td>
<td>Any</td>
<td>Switches the weapon fire mode</td>
</tr>
</tbody>
</table>

FireWeapon node

Use to fire a weapon and set a target entity or a target position.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetId</td>
<td>Any</td>
<td>Target ID</td>
</tr>
<tr>
<td>TargetPos</td>
<td>Vec3</td>
<td>Target position</td>
</tr>
<tr>
<td>AlignToTarget</td>
<td>Boolean</td>
<td>Aims the weapon at the target</td>
</tr>
<tr>
<td>StartFire</td>
<td>Any</td>
<td>Starts firing weapon</td>
</tr>
<tr>
<td>StopFire</td>
<td>Any</td>
<td>Stops firing weapon</td>
</tr>
<tr>
<td>NumberOfShots</td>
<td>Integer</td>
<td>Fires the specified number of shots</td>
</tr>
<tr>
<td>Accuracy</td>
<td>Float</td>
<td>Specifies firing accuracy from 0% to 100%</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FireStarted</td>
<td>Boolean</td>
<td>Triggers when firing starts</td>
</tr>
</tbody>
</table>
### XML Nodes

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FireStopped</td>
<td>Boolean</td>
<td>Triggers when firing stops</td>
</tr>
</tbody>
</table>

**Listener node**

Use to listen on WeaponId or player's WeaponClass, or as a fallback on the current player's weapon and to trigger OnShoot when shot.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Any</td>
<td>Enable listener</td>
</tr>
<tr>
<td>Disable</td>
<td>Any</td>
<td>Disables listener</td>
</tr>
<tr>
<td>WeaponId</td>
<td>Any</td>
<td>Weapon ID</td>
</tr>
<tr>
<td>WeaponClass</td>
<td>String</td>
<td>Weapon name</td>
</tr>
<tr>
<td>ShootCount</td>
<td>Integer</td>
<td>Number of times the listener can be triggered. 0 = infinite</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnShoot</td>
<td>Any</td>
<td>Triggered when shooting</td>
</tr>
<tr>
<td>ShootsLeft</td>
<td>Integer</td>
<td>Triggered when shooting left</td>
</tr>
<tr>
<td>OnMelee</td>
<td>Any</td>
<td>Triggered on melee attack</td>
</tr>
<tr>
<td>OnDropped</td>
<td>String</td>
<td>Triggered when weapon is dropped</td>
</tr>
</tbody>
</table>

**XML Nodes**

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the following flow graph nodes to specify XML elements.
Topics
- ClearValue node (p. 676)
- DeleteAllAttributes node (p. 677)
- DeleteAllChildren node (p. 677)
- DeleteAttribute node (p. 677)
- DeleteChild node (p. 678)
- DeleteChildAt node (p. 678)
- GetAttribute node (p. 679)
- GetAttributeCount node (p. 679)
- GetChild node (p. 680)
- GetChildAt node (p. 680)
- GetChildCount node (p. 681)
- GetParent node (p. 681)
- GetRoot node (p. 682)
- GetValue node (p. 682)
- HasAttribute node (p. 683)
- IncAttribute node (p. 683)
- IncValue node (p. 684)
- NewChild node (p. 684)
- NewDocument node (p. 685)
- OpenDocument node (p. 685)
- SaveDocument node (p. 686)
- SetAttribute node (p. 686)
- SetValue node (p. 687)

ClearValue node

Used to clear the value of the active element.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>Executes the instruction</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>Called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>Called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Called when the instruction has completed carrying out</td>
</tr>
</tbody>
</table>
**DeleteAllAttributes node**

Used to delete all attributes from the active element.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>Executes the instruction</td>
</tr>
</tbody>
</table>

**Inputs**

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>Called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>Called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Called when the instruction has completed carrying out</td>
</tr>
</tbody>
</table>

**DeleteAllChildren node**

Used to delete all children of the active element.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>Executes the instruction</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>Optional child name</td>
</tr>
</tbody>
</table>

**Inputs**

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>Called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>Called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Called when the instruction has completed carrying out</td>
</tr>
</tbody>
</table>

**DeleteAttribute node**

Used to delete an attribute from the active element.
XML Nodes

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>executes the instruction</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>optional child name</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>called when the instruction has completed carrying out</td>
</tr>
</tbody>
</table>

DeleteChild node

Used to delete the first child node with the given name.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>executes the instruction</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>optional child name</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>called when the instruction has completed carrying out</td>
</tr>
</tbody>
</table>

DeleteChildAt node

Used to delete the nth child node with the given name.
Lumberyard Legacy Reference
XML Nodes

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>Executes the instruction</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>Name of the attribute</td>
</tr>
<tr>
<td>Index</td>
<td>Integer</td>
<td>Location of the child node in the list</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>Called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>Called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Called when the instruction has completed carrying out</td>
</tr>
</tbody>
</table>

GetAttribute node

Used to get the value of an attribute for the active element.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>Executes the instruction</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>Name of the attribute</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>Called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>Called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Called when the instruction has completed carrying out</td>
</tr>
</tbody>
</table>

GetAttributeCount node

Used to get the number of attributes for the active element.
**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>Executes the instruction</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>Called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>Called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Called when the instruction has completed carrying out</td>
</tr>
<tr>
<td>Count</td>
<td>Integer</td>
<td>Outputs the count</td>
</tr>
</tbody>
</table>

**GetChild node**

Used to navigate to the first child node with the given name.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>Executes the instruction</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>Name of the attribute</td>
</tr>
<tr>
<td>Create</td>
<td>Boolean</td>
<td>Creates a child node if one does not exist</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>Called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>Called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Called when the instruction has completed carrying out</td>
</tr>
</tbody>
</table>

**GetChildAt node**

Used to navigate to the nth child node with the given name.
**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>Executes the instruction</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>Name of the child node</td>
</tr>
<tr>
<td>Index</td>
<td>Integer</td>
<td>Location of the child node in the list</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>Called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>Called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Called when the instruction has completed carrying out</td>
</tr>
</tbody>
</table>

**GetChildCount node**

Used to return the number of children of the active element.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>Executes the instruction</td>
</tr>
</tbody>
</table>

**GetParent node**

Used to sets the active element to the current active element's parent (move one up).
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>Executes the instruction</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>Called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>Called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Called when the instruction has completed carrying out</td>
</tr>
</tbody>
</table>

GetRoot node

Used to set the active element to the root node (move to top).

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>Executes the instruction</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>Called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>Called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Called when the instruction has completed carrying out</td>
</tr>
</tbody>
</table>

GetValue node

Used to get the value of the active element.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>Executes the instruction</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>Called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>Called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Called when the instruction has completed carrying out</td>
</tr>
<tr>
<td>Value</td>
<td>Any</td>
<td>Outputs the value of the element</td>
</tr>
</tbody>
</table>

**HasAttribute node**

Used to check if an attribute exists for the active element.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>Enables the instruction</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>Name of the attribute</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>Called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>Called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Called when the instruction has completed carrying out</td>
</tr>
<tr>
<td>Yes</td>
<td>Any</td>
<td>Has the attribute</td>
</tr>
<tr>
<td>No</td>
<td>Any</td>
<td>Does not have the attribute</td>
</tr>
<tr>
<td>Result</td>
<td>Boolean</td>
<td>Boolean result</td>
</tr>
</tbody>
</table>

**IncAttribute node**

Used to increment an attribute by the given amount for the active element.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>Executes the instruction</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>Name of the attribute</td>
</tr>
<tr>
<td>Amount</td>
<td>Float</td>
<td>Amount to increment by</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>Called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>Called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Called when the instruction has completed carrying out</td>
</tr>
</tbody>
</table>

**IncValue node**

Used to increment the value of the active element.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>Executes the instruction</td>
</tr>
<tr>
<td>Amount</td>
<td>Float</td>
<td>Amount to increment by</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>Called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>Called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Called when the instruction has completed carrying out</td>
</tr>
</tbody>
</table>

**NewChild node**

Used to create a new child node at end of parent's sibling list.
# XML Nodes

## Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>Executes the instruction</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>Name of the child node</td>
</tr>
<tr>
<td>Active</td>
<td>Boolean</td>
<td>Makes the child node the active element</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>Called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>Called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Called when the instruction has completed carrying out</td>
</tr>
</tbody>
</table>

## NewDocument node

Used to create a blank document for writing new data into.

```
<NewDocument>
  <Execute>
    <Success>
    <Root>Root</Root>
  <Done>
</NewDocument>
```

## Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>Executes the instruction</td>
</tr>
<tr>
<td>Root</td>
<td>String</td>
<td>Name of the XML root element</td>
</tr>
</tbody>
</table>

## Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>Called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>Called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Called when the instruction has completed carrying out</td>
</tr>
</tbody>
</table>

## OpenDocument node

Used to open an XML document from disk.

```
<OpenDocument>
  <Execute>
    <Success>
    <File>default</File>
  <Done>
</OpenDocument>
```
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>Executes the instruction</td>
</tr>
<tr>
<td>File</td>
<td>String</td>
<td>File name of the XML document</td>
</tr>
<tr>
<td>Location</td>
<td>Integer</td>
<td>File path of the XML document</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>Called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>Called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Called when the instruction has completed carrying out</td>
</tr>
</tbody>
</table>

SaveDocument node

Used to save active XML data to disk.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>Executes the instruction</td>
</tr>
<tr>
<td>File</td>
<td>String</td>
<td>File name of the saved XML document</td>
</tr>
<tr>
<td>Location</td>
<td>Integer</td>
<td>File path of the XML document</td>
</tr>
<tr>
<td>Overwrite</td>
<td>Boolean</td>
<td>Determines where document should overwrite existing XML document</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>Called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>Called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Called when the instruction has completed carrying out</td>
</tr>
</tbody>
</table>

SetAttribute node

Used to set an attribute for the active element.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>Executes the instruction</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>Name of the attribute to set</td>
</tr>
<tr>
<td>Value</td>
<td>Any</td>
<td>Sets the value of the element</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>Called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>Called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Called when the instruction has completed carrying out</td>
</tr>
</tbody>
</table>

**Set Value node**

Used to set the value of the active element.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td>Any</td>
<td>Executes the instruction</td>
</tr>
<tr>
<td>Value</td>
<td>Any</td>
<td>Sets the value of the attribute</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Boolean</td>
<td>Called if the instruction executed successfully</td>
</tr>
<tr>
<td>Fail</td>
<td>Boolean</td>
<td>Called if the instruction failed</td>
</tr>
<tr>
<td>Done</td>
<td>Boolean</td>
<td>Called when the instruction has completed carrying out</td>
</tr>
</tbody>
</table>
Using Flow Graph Links

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

Links are used to connect Flow Graph node inputs and outputs for transferring information between them. Information is transferred immediately, regardless of link length or shape. When a connected node is moved, the link automatically adjusts itself. Links are created by simply clicking and dragging your mouse from the output of one node to the input of another node.

An input port can have only one link connected to it. If you want to connect multiple links to one input port, helper nodes such as the Logic:Any node can be used. Output ports can have an unlimited number of links.

Node links can be deleted or disabled. If you merely want to disable a link but still have it show on the flow graph, click Disable instead.

To delete a node link

1. Click the link to select it, right-click the dot in the middle of the link, then click Remove.
2. Alternatively, click the input port the link is connected to and drag it away from the port. When the mouse is released, the link disappears.

By default, all information between nodes is relayed instantly. However, you can delay signal propagation between nodes.

To delay link propagation

1. Click the link to select it, right-click the dot in the middle of the link, then click Delay.
2. In the new Time:Delay node, double-click Delay and enter a value in seconds. The default value is 1 second if no value is entered.

Connecting multiple links to an input port is possible using the Logic:Any node. This node can take multiple inputs and route the signals to a single output port.
To add multiple links to an input port

1. Right-click anywhere in the graph pane and then click Add node, Logic, Any.
2. Drag from the various output port links to the in1...in10 input ports of the Logic:Any node as needed.
3. Create links by dragging from the out port to the input ports of the desired nodes.

You can also highlight links to make debugging complex flow graphs easier.

- To highlight incoming links red, select an input node and press F.
- To highlight outgoing links blue, select an output node and press G.

Using Flow Graph Tokens

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard's new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

A flow graph token is a variable used for storing values for reuse in the same flow graph. Flow graph tokens can be used for performing simple logic and checks within a flow graph script. They are typically used to send different variables across a very large flow graph and to alleviate the need for extra node links.

Flow graph tokens share many similarities with game tokens. They can have the same types of variables set and even appear under the command `gt_show=1` along with the rest of the game tokens.
To create a Flow Graph token

1. In Flow Graph Editor, click **Tools, Edit Graph Tokens**.
2. In the **Graph Tokens** window, click **New Token**, then name the token.
3. Right-click anywhere in the flow graph, then click **Add Node, Mission, GameTokenSet**.
4. In the **Mission:GameTokenSet** node, double-click **Value** and enter a value.

Managing Flow Graph Modules

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about **Script Canvas**, Lumberyard’s new visual scripting environment.

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 about the latest features, see the **Amazon Lumberyard User Guide**.

A module is simply an exported flow graph that can be loaded and called from another flow graph during gameplay.

Any flow graph can be converted to a module by first creating a new module using Flow Graph Editor and then copying the flow graph contents to the new module.

Modules used in multiple levels are called global modules, while modules used only in a specific level are called level modules.

The advantages of using modules include:

- Flow graphs can be used in multiple levels, but exist in a single location
- Modules can receive unique input values from their callers, allowing them to be robust
• Modules can return unique output values to their callers, allowing them to be used in different situations
• Modules can be instanced, so multiple copies of the same module can be active simultaneously, but running with different inputs

To create or delete a module

In Flow Graph Editor, under Flow Graphs do the following:

1. To create a module, right click FG Modules, then click New Global FG Module or New Level FG Module as applicable. The new module appears under the Global or Local folders respectively.
2. To delete a module, right-click the module and click Delete Module.

Module Node Ports

Flow Graph Module nodes have a variety of different of input and output node ports.

All inputs passed to the Call node activates the corresponding outputs on the Start node, and similarly inputs to the End node passes back to the Call node when Success or Cancel are activated.

Module Inputs

• Call - Call to load and start the module. If the module is already started it triggers the update port of the Start node with updated parameters if not instanced. It is named Module:Call_YourModuleName.
• Instanced - If set to 1 (default), creates a new independent instance of the module whenever you trigger the Call input port.
• Cancel - Cancels the module. This requires the correct InstanceID if instanced.
• InstanceID - Identifies a module instance. A value of -1 (default) creates a new instance; otherwise, it updates the given instance if instanced.

Module Outputs

• OnCalled - Called when module is started. Returns a value of -1 if the module is not instanced.
• Done - Called when the module returns with a success status.
• Canceled - Called when the module returns with a failed status.

You can also customize the inputs and outputs for each module to pass extra data back and forth.

To customize module ports

1. In Flow Graph Editor, select the module, then click Tools, Edit Module.
2. In the Module Ports dialog box, click Edit Input or Edit Output as needed, then make a Type selection as follows:
   • Bool
   • EntityId
   • Int
   • Float
   • String
   • Vec3
3. Click **OK** to update module nodes with the changes.

## Debugging Flow Graph

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about [Script Canvas](https://developer.amazon.com/canvas), Lumberyard's new visual scripting environment.

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](https://docs.aws.amazon.com/lumberyard/latest/ug/configurator.html) or the command line. To learn about the latest features, see the [Amazon Lumberyard User Guide](https://developer.amazon.com/lumberyard).

### Topics
- Using Flow Graph Debugger (p. 692)
- Using Console Variables (p. 693)

## Using Flow Graph Debugger

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about [Script Canvas](https://developer.amazon.com/canvas), Lumberyard's new visual scripting environment.

Using the Flow Graph Debugger, you can add breakpoints to any input or output port of a node. Once a node port is triggered, the game is paused and the Flow Graph Editor displays the applicable node in the center of the graph pane.

To enable Flow Graph Debugger, click the bug (toggle visual flowgraph debugging) toolbar icon in Flow Graph Editor.

To resume the game once a breakpoint is triggered, click the play (Start Flowgraph Update) toolbar icon, or press **F5**.

### To manage Flow Graph breakpoints

- In Flow Graph Editor, right-click the applicable input or output node port, then do the following as needed:
  - To create a breakpoint, click **Add Breakpoint**. A red dot is displayed next to the node port.
  - To remove a breakpoint, click **Remove Breakpoint**.
  - To enable or disable a breakpoint, toggle the **Enabled** check box.
  - To remove all breakpoints on a node, or for all nodes on the entire flow graph, click **Remove Breakpoints for Node** or **Remove Breakpoints for Graph** respectively.
Every breakpoint can be converted to a tracepoint, which instead of pausing the game outputs the information about a triggered breakpoint to the console and to a log file. Simply right-click on the applicable breakpoint-enabled node port, then click Tracepoint. The red dot changes to a red diamond to indicate that the port has a tracepoint enabled on it.

Tracepoint data sent to the Console looks like this, as an example:

```
[TRACEPOINT HIT - FrameID: 71054] GRAPH: AnimObject1 (ID: 96) - NODE:
  Entity:MaterialParam (ID: 5) - PORT: ValueColor - VALUE:
  0.867136,0.005522,0.005522
```

### Using Console Variables

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard’s new visual scripting environment.

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 about the latest features, see the Amazon Lumberyard User Guide.

The following console variables can be used to troubleshoot Flow Graph issues.

- `fg_abortOnLoadError` — Aborts on a loading error of a flow graph, where 0=dialog, 1=log, 2=abort
- `fg_debugmodules` — 0=disabled, 1=show all modules, 2=show all modules and active modules
- `fg_debugmodules_filter filterstring` — Used to only show modules that match the filterstring
- `fg_iDebugNextStep` — Step-by-step debugging
- `fg_iEnableFlowgraphNodeDebugging` — toggles flow graph debugging of nodes
- `fg_inspectorLog` — log inspector on Console
- `fg_noDebugText` — Don’t display flow graph debugging text
- `fg_profile` — toggles flow graph profiling
- `fg_SystemEnable` — toggles Flow Graph system updates
- `gt_showFilter` — Filter string for flow graph tokens and game tokens
- `gt_showLines` — Specifies how many lines to display
- `gt_showPosX 0` — Shows the x-axis position
- `gt_showPosY` — Shows the y-axis position
- `gt_show Value` — Shows game token and graph token state, where 1=screen and log, 2=screen only, 3=log only.

### Placing Cached Shadows

Flow Graph is deprecated and will be removed in a future version of Lumberyard. Learn about Script Canvas, Lumberyard’s new visual scripting environment.
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 about the latest features, see the Amazon Lumberyard User Guide.

Cached shadows display shadow properties for an entire scene. It replaces the shadow cascades that appear farthest from the viewer and reduces the number of shadow draw calls per frame.

**Note**
To eliminate visible artifacts when time of day is updated or in scenes that have huge objects casting shadows in the distance, we recommend turning off cached shadows.

To specify placement of cached shadows, use the Flow Graph Editor. Before you trigger an update, compile all of your shaders to ensure that all objects are rendered into the cached shadows.

Use `Environment:RecomputeStaticShadows` for cached shadows. This node takes the minimum and maximum positions of the world space bounding area, and triggers the re-rendering of the cached shadows.

**Recommended Settings**

**r_ShadowsCache**

Default value: 4

Bounding area: 1000 x 1000 meters (recommended maximum, X/Y direction) and as small a range as possible (Z direction)

**Related Console Variables**

**r_ShadowsCache**

Replaces all sun cascades above the specified console variable (cvar) value with cached shadows.

Valid values: 0=no cached shadows | 1=replace first cascade and up | 2=replace second cascade and up | etc.

**r_ShadowsStaticMapResolution**

The resolution of the cached shadows. The cached shadows for mobile platforms has 16 bit precision and consumes 8 MB of video memory. The cached shadows for other platforms has 16 bit precision and consumes 128 MB of video memory.

Default value: 2048 (mobile platforms), 8192 (other platforms)

**e_ShadowsStaticMapUpdate**

Triggers update of the cached shadows.

Valid values: 0=no update | 1=one update | 2=continuous updates

**e_ShadowsStaticObjectLod**

The level of detail (LOD) used for rendering objects into the cached shadows.
Cloud Canvas Flow Graph Node 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 about the latest features, see the Amazon Lumberyard User Guide.

This section provides a legacy reference for the flow graph nodes in Cloud Canvas.

- Cloud Canvas Configuration Nodes (p. 695)
- Cognito (Player Identity) Nodes (p. 698)
- DynamoDB (Database) Nodes (p. 700)
- Lambda (Cloud Functions) Node (p. 707)
- S3 (Storage) Nodes (p. 708)
- SNS (Notification Service) Nodes (p. 710)
- SQS (Message Queuing Service) Nodes (p. 713)
- Static Data (PROTOTYPE) Nodes (p. 714)

Note
The use of flow graph nodes for Cloud Canvas is deprecated. Instead, see cloud gems in the Amazon Lumberyard User Guide to implement cloud-connected features in your games.

To expose AWS services like Amazon API Gateway, AWS Lambda, and Amazon S3 to script, you can use the CloudGemAWS::ScriptBehaviors gem.

Topics
- Cloud Canvas Configuration Nodes (p. 695)
- Cognito (Player Identity) Nodes (p. 698)
- DynamoDB (Database) Nodes (p. 700)
- Lambda (Cloud Functions) Node (p. 707)
- S3 (Storage) Nodes (p. 708)
- SNS (Notification Service) Nodes (p. 710)
- SQS (Message Queuing Service) Nodes (p. 713)
- Static Data (PROTOTYPE) Nodes (p. 714)

Cloud Canvas Configuration Nodes

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use these flow graph nodes to configure Cloud Canvas settings.

Topics
- ApplyConfiguration node (p. 696)
- SetConfigurationVariable node (p. 696)
ApplyConfiguration node
Applies AWS configuration to all managed clients.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apply</td>
<td>Any</td>
<td>Applies the current AWS configuration to all managed clients</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
</tbody>
</table>

SetConfigurationVariable node
Sets a configuration parameter value.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure</td>
<td>Any</td>
<td>Sets the parameter value</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>Name of the parameter to set</td>
</tr>
<tr>
<td>Value</td>
<td>String</td>
<td>Value to which the parameter will be set; may contain $param-name$ substrings</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
</tbody>
</table>
ConfigureProxy node

Sets the proxy configuration used by all AWS clients.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure</td>
<td>Any</td>
<td>Sets the proxy configuration</td>
</tr>
<tr>
<td>Host</td>
<td>String</td>
<td>Proxy host</td>
</tr>
<tr>
<td>Port</td>
<td>Integer</td>
<td>Proxy port</td>
</tr>
<tr>
<td>UserName</td>
<td>String</td>
<td>Proxy user name</td>
</tr>
<tr>
<td>Password</td>
<td>String</td>
<td>Proxy password</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
</tbody>
</table>

GetConfigurationVariableValue node

Inserts configuration value parameters into a string.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expand</td>
<td>Any</td>
<td>Expands parameter references</td>
</tr>
<tr>
<td>Value</td>
<td>String</td>
<td>Value containing $param-name$ substrings</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
</tbody>
</table>
Cognito (Player Identity) Nodes

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
<tr>
<td>Value</td>
<td>String</td>
<td>Value with $param-name$ substring replaced by parameter values</td>
</tr>
</tbody>
</table>

**SetDefaultRegion node**

Sets (overrides) the region for all AWS clients in the current project.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>Any</td>
<td>Sets the region for all AWS clients in the current project</td>
</tr>
<tr>
<td>Region</td>
<td>String</td>
<td>The region name to set as the default region for all AWS clients</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
</tbody>
</table>

Choose **Apply** if you want to apply the configuration change to all AWS clients immediately. If **Apply** is set to **false**, you must add an **ApplyConfiguration** (p. 696) flow node to activate the changes.

**Cognito (Player Identity) Nodes**

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 about the latest features, see the Amazon Lumberyard User Guide.

Use Amazon Cognito to configure player identity with these flow graph nodes.

**Topics**
- ConfigureAnonymousPlayer node (p. 699)
- ConfigureAuthenticatedPlayer node (p. 699)
ConfigureAnonymousPlayer node

Creates an anonymous identity on the device in your AWS account.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure</td>
<td>Any</td>
<td>Configure your game to use Amazon Cognito for anonymous players</td>
</tr>
<tr>
<td>AWSAccountNumber</td>
<td>String</td>
<td>Your AWS account number. This is needed to access Amazon Cognito.</td>
</tr>
<tr>
<td>IdentityPoolID</td>
<td>String</td>
<td>The unique ID of your Amazon Cognito identity pool. To create an identity pool ID, sign in to the AWS Management Console and use the Amazon Cognito console at <a href="https://console.aws.amazon.com/cognito/">https://console.aws.amazon.com/cognito/</a>.</td>
</tr>
<tr>
<td>CachingFileLocationOverride</td>
<td>String</td>
<td>If specified, causes the Amazon Cognito ID to be cached to the path specified instead of to <code>HOME_DIR/.aws/.identities</code>.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
<tr>
<td>CognitoIdentityID</td>
<td>String</td>
<td>The unique ID of the user</td>
</tr>
</tbody>
</table>

The first time the player runs the game and this node is triggered, an anonymous ID is generated for the player. This ID is persisted locally, and future runs of the game use the same identity.

ConfigureAuthenticatedPlayer node

Creates an authenticated identity on the device in your AWS account.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure</td>
<td>Any</td>
<td>Configure your game to use Amazon Cognito with the values specified.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
<tr>
<td>CognitoIdentityID</td>
<td>String</td>
<td>The unique ID of the user</td>
</tr>
<tr>
<td>Port</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AWSAccountNumber</td>
<td>String</td>
<td>Your AWS account number. This is needed for configuring Amazon Cognito.</td>
</tr>
<tr>
<td>IdentityPoolID</td>
<td>String</td>
<td>The unique ID of your Amazon Cognito identity pool. To edit your identity pool ID, open the AWS Management Console and choose Cognito.</td>
</tr>
<tr>
<td>ProviderName</td>
<td>String</td>
<td>Specifies the provider that authenticates the user</td>
</tr>
<tr>
<td>ProviderToken</td>
<td>String</td>
<td>Provider token with which to authenticate the user</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
<tr>
<td>CognitoIdentityID</td>
<td>String</td>
<td>The unique ID of the user</td>
</tr>
</tbody>
</table>

The first time the player runs the game and this node is triggered, an authenticated ID is generated for the player. The same ID is returned any time the user logs in with the same account, even on a second device.

**DynamoDB (Database) Nodes**

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use these flow graph nodes to connect your game to Amazon DynamoDB.

**Topics**

- AtomicAdd node (p. 700)
- DeleteItem node (p. 701)
- GetItem node (p. 702)
- PutItem node (p. 703)
- Query node (p. 704)
- ScanTable node (p. 704)
- UpdateItem node (p. 705)
- GetStringSet node (p. 706)

**AtomicAdd node**

Add a number to an attribute in DynamoDB and return the number.
Lumberyard Legacy Reference
DynamoDB (Database) Nodes

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Any</td>
<td>Writes the value specified in the Value port to DynamoDB.</td>
</tr>
<tr>
<td>TableName</td>
<td>String</td>
<td>The name of the DynamoDB table to which to write.</td>
</tr>
<tr>
<td>TableKeyName</td>
<td>String</td>
<td>The key name used in the table.</td>
</tr>
<tr>
<td>Key</td>
<td>String</td>
<td>Specifies the key to which to write.</td>
</tr>
<tr>
<td>Attribute</td>
<td>String</td>
<td>Specifies the attribute to which to write.</td>
</tr>
<tr>
<td>Value</td>
<td>Integer</td>
<td>Specifies the value to write.</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation.</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message.</td>
</tr>
<tr>
<td>NewValue</td>
<td>String</td>
<td>The value of the attribute after the addition.</td>
</tr>
</tbody>
</table>

This is an atomic operation. You do not need to create the attribute before you use it.

**DeleteItem node**

Deletes a record in DynamoDB.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeleteItem</td>
<td>Any</td>
<td>Deletes the specified item from DynamoDB.</td>
</tr>
<tr>
<td>TableName</td>
<td>String</td>
<td>The name of the DynamoDB table from which to delete.</td>
</tr>
<tr>
<td>TableKeyName</td>
<td>String</td>
<td>The key name used in the table.</td>
</tr>
</tbody>
</table>
## DynamoDB (Database) Nodes

### Port

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyValue</td>
<td>String</td>
<td>Specifies the key to delete</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
<tr>
<td>DeletedItems</td>
<td>Any</td>
<td>Activated when matches were found to delete</td>
</tr>
<tr>
<td>NoResults</td>
<td>Any</td>
<td>No matching results were found</td>
</tr>
</tbody>
</table>

## GetItem node

Gets values from DynamoDB.

![Cloud Canvas (AWS): AWS Services: DynamoDB (Database): GetItem node](image)

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetItem</td>
<td>Any</td>
<td>Retrieves the item specified from DynamoDB</td>
</tr>
<tr>
<td>TableName</td>
<td>String</td>
<td>The name of the DynamoDB table from which to read</td>
</tr>
<tr>
<td>TableKeyName</td>
<td>String</td>
<td>The key name used in the table</td>
</tr>
<tr>
<td>KeyValue</td>
<td>String</td>
<td>Specifies the key to read</td>
</tr>
<tr>
<td>AttributeToReturn</td>
<td>String</td>
<td>Specifies the attribute to read</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
<tr>
<td>DataOut</td>
<td>String</td>
<td>String data that was read from DynamoDB</td>
</tr>
<tr>
<td>NumberOut</td>
<td>String</td>
<td>Number data that was read from DynamoDB</td>
</tr>
<tr>
<td>BoolOut</td>
<td>String</td>
<td>Boolean value that was read from DynamoDB</td>
</tr>
</tbody>
</table>
### PutItem node

**Writes values to DynamoDB.**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PutItem</td>
<td>Any</td>
<td>Writes the item specified to DynamoDB</td>
</tr>
<tr>
<td>TableName</td>
<td>String</td>
<td>The name of the DynamoDB table to which to write</td>
</tr>
<tr>
<td>TableKeyName</td>
<td>String</td>
<td>The key name used in the table</td>
</tr>
<tr>
<td>KeyValue</td>
<td>String</td>
<td>Specifies the key to write</td>
</tr>
<tr>
<td>AttributeToWrite</td>
<td>String</td>
<td>Specifies the attribute to write</td>
</tr>
<tr>
<td>DataIn</td>
<td>String</td>
<td>The data to write</td>
</tr>
<tr>
<td>DataType</td>
<td>String</td>
<td>The data type that the data will be written as</td>
</tr>
<tr>
<td>KeyMustNotExist</td>
<td>Boolean</td>
<td>When true, specifies that the key must not already exist; the default is true. Setting this to false allows you to overwrite an existing key in the table, including all of its existing attributes, and replace them with the new key and attribute values.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
<tr>
<td>KeyAlreadyExists</td>
<td>Any</td>
<td>The key already exists; no change was made</td>
</tr>
</tbody>
</table>
Query node

Queries values in DynamoDB.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td>Any</td>
<td>Queries table data in DynamoDB</td>
</tr>
<tr>
<td>TableName</td>
<td>String</td>
<td>The name of the DynamoDB table to query</td>
</tr>
<tr>
<td>TableKeyName</td>
<td>String</td>
<td>The name of the table key to query</td>
</tr>
<tr>
<td>KeyValue</td>
<td>String</td>
<td>The value of the key to query</td>
</tr>
<tr>
<td>AttributeToCheck</td>
<td>String</td>
<td>The attribute to query</td>
</tr>
<tr>
<td>AttributeComparisonType</td>
<td>String</td>
<td>The comparison type to make against the attribute; the default is EQUALS. Other possible values are GREATER_THAN, GREATER_THAN_OR_EQUALS, LESS_THAN, LESS_THAN_OR_EQUALS.</td>
</tr>
<tr>
<td>AttributeComparisonValue</td>
<td>String</td>
<td>The value to compare against the attribute</td>
</tr>
<tr>
<td>AttributeComparisonValueType</td>
<td>String</td>
<td>The data type of AttributeComparisonValue (string, bool, or number); the default is string</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
<tr>
<td>MatchFound</td>
<td>Any</td>
<td>A match was found</td>
</tr>
<tr>
<td>NoMatch</td>
<td>Any</td>
<td>No match was found</td>
</tr>
</tbody>
</table>

ScanTable node

Scans for entries which pass a comparison test in DynamoDB.
Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan</td>
<td>Any</td>
<td>Scans for matches in DynamoDB table data using the specified attributes</td>
</tr>
<tr>
<td>TableName</td>
<td>String</td>
<td>The name of the DynamoDB table to scan</td>
</tr>
<tr>
<td>Attribute</td>
<td>String</td>
<td>The attribute to query for</td>
</tr>
<tr>
<td>AttributeComparisonType</td>
<td>String</td>
<td>The comparison type to make against the attribute; this defaults to EQUALS.</td>
</tr>
<tr>
<td>AttributeComparisonValue</td>
<td>String</td>
<td>The value to compare against the attribute</td>
</tr>
<tr>
<td>AttributeComparisonValueType</td>
<td>String</td>
<td>The data type of AttributeComparisonValue (string, bool, or number); the default is string</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
<tr>
<td>MatchesFound</td>
<td>Any</td>
<td>The number of matches found on a successful scan</td>
</tr>
</tbody>
</table>

UpdateItem node

Updates attribute values of an existing item in DynamoDB.
### GetStringSet Node

Retrieves the members of a string set.

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetItem</td>
<td>Any</td>
<td>Reads data from DynamoDB</td>
</tr>
<tr>
<td>TableName</td>
<td>String</td>
<td>The name of the DynamoDB table to use</td>
</tr>
<tr>
<td>TableKeyName</td>
<td>String</td>
<td>The name of the key in the table</td>
</tr>
<tr>
<td>Key</td>
<td>String</td>
<td>The value of the key to write</td>
</tr>
<tr>
<td>AttributeToWrite</td>
<td>String</td>
<td>The attribute to write to</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
<tr>
<td>ConditionsFailed</td>
<td>String</td>
<td>Key or attribute not found (either the KeyMustExist or AttributeMustExist condition failed)</td>
</tr>
</tbody>
</table>
### Lambda (Cloud Functions) Node

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use this flow graph node to invoke AWS Lambda functions.

#### Invoke node

![Invoke node diagram]

#### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invoke</td>
<td>Any</td>
<td>Invokes a Lambda function, optionally providing JSON data as arguments through the Args port. For more information, see AWS Lambda Invoke Request Syntax.</td>
</tr>
<tr>
<td>FunctionName</td>
<td>String</td>
<td>The name of the Lambda function to call</td>
</tr>
<tr>
<td>Args</td>
<td>String</td>
<td>The input data that will be sent to the Lambda function call as arguments in JSON format. For more information, see AWS Lambda Invoke Request Syntax.</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
<tr>
<td>Result</td>
<td>String</td>
<td>The data that was output by the Lambda function if no error occurred</td>
</tr>
</tbody>
</table>

The success port indicates that all members of the set have been output.

### Lumberyard Legacy Reference

#### Lambda (Cloud Functions) Node

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
<tr>
<td>DataOut</td>
<td>String</td>
<td>The data read from DynamoDB. The DataOut port fires once for each member of the set.</td>
</tr>
</tbody>
</table>

The success port indicates that all members of the set have been output.
S3 (Storage) Nodes

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use these flow graph nodes to download and upload files from the Amazon Simple Storage Service (Amazon S3), and to generate a public URL that points to a specific location in Amazon S3.

Topics

- DownloadFile node (p. 708)
- UploadFile node (p. 708)
- GeneratePublicUrl node (p. 709)

**DownloadFile node**

Downloads a file from Amazon S3.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DownloadFile</td>
<td>Any</td>
<td>Reads file data from an Amazon S3 bucket</td>
</tr>
<tr>
<td>BucketName</td>
<td>String</td>
<td>The name of the Amazon S3 bucket to use</td>
</tr>
<tr>
<td>KeyName</td>
<td>String</td>
<td>The name of the file to download from Amazon S3</td>
</tr>
<tr>
<td>FileName</td>
<td>String</td>
<td>The filename to use for the downloaded object</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
</tbody>
</table>

**UploadFile node**

Uploads a file to Amazon S3.
Lumberyard Legacy Reference
S3 (Storage) Nodes

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UploadFile</td>
<td>Any</td>
<td>Uploads a file to an Amazon S3 bucket</td>
</tr>
<tr>
<td>BucketName</td>
<td>String</td>
<td>The name of the Amazon S3 bucket to use</td>
</tr>
<tr>
<td>KeyName</td>
<td>String</td>
<td>What to name the uploaded object on Amazon S3. If this value is not updated</td>
</tr>
<tr>
<td></td>
<td></td>
<td>on subsequent uses, the existing Amazon S3 object is overwritten.</td>
</tr>
<tr>
<td>Content-Type</td>
<td>String</td>
<td>The mime-content type to use for the uploaded object (for example, text/html,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>video/mpeg, video/avi, or application/zip). The type is stored in the Amazon</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S3 record. You can use this type to help identify or retrieve a specific type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of data later. The default is application/octet-stream.</td>
</tr>
<tr>
<td>FileName</td>
<td>String</td>
<td>The name of the file to upload</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outputs</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>message</td>
</tr>
</tbody>
</table>

**GeneratePublicUrl node**

Generates a presigned URL that points to an Amazon S3 location that you specify.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PresignUrl</td>
<td>Any</td>
<td>Generates a presigned URL for the Amazon S3 location specified</td>
</tr>
<tr>
<td>BucketName</td>
<td>String</td>
<td>The name of the Amazon S3 bucket to use</td>
</tr>
</tbody>
</table>
### SNS (Notification Service) Nodes

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 about the latest features, see the Amazon Lumberyard User Guide.

You can use these flow graph nodes to process Amazon Simple Notification Service (Amazon SNS) messages.

**Topics**
- ParseMessage node (p. 710)
- Notify node (p. 711)
- CheckArnSubscribed node (p. 711)
- SubscribeToTopic node (p. 712)

#### ParseMessage node

![Diagram of ParseMessage node]

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>Any</td>
<td>Extract the subject and body text in JSON format from an Amazon SNS message</td>
</tr>
<tr>
<td>Message</td>
<td>String</td>
<td>The JSON message to deserialize.</td>
</tr>
</tbody>
</table>

---

### Port | Type | Description
---|---|---
KeyName | String | What to name the uploaded object on Amazon S3. If this value is not updated on subsequent uses, the existing Amazon S3 object is overwritten.
Http Request Method | String | The HTTP method against which to presign (DELETE, GET, POST, or PUT)
Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
<tr>
<td>Body</td>
<td>String</td>
<td>The message body</td>
</tr>
<tr>
<td>Subject</td>
<td>String</td>
<td>The message subject</td>
</tr>
</tbody>
</table>

Notify node

Publishes messages to an Amazon SNS topic.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notify</td>
<td>Any</td>
<td>Sends a notification to an Amazon SNS topic</td>
</tr>
<tr>
<td>Message</td>
<td>String</td>
<td>The message to send</td>
</tr>
<tr>
<td>Subject</td>
<td>String</td>
<td>The subject of the message</td>
</tr>
<tr>
<td>TopicARN</td>
<td>String</td>
<td>The Amazon Resource Name for your Amazon SNS topic</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
</tbody>
</table>

CheckArnSubscribed node

Checks if an ARN is subscribed to an Amazon SNS topic.

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check</td>
<td>Success</td>
<td></td>
</tr>
<tr>
<td>TopicARN</td>
<td>Error</td>
<td></td>
</tr>
<tr>
<td>Endpoint</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td></td>
<td>False</td>
<td></td>
</tr>
</tbody>
</table>
### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check</td>
<td>Any</td>
<td>Checks if an ARN is subscribed to an Amazon SNS topic</td>
</tr>
<tr>
<td>TopicARN</td>
<td>String</td>
<td>The Amazon SNS topic ARN to check</td>
</tr>
<tr>
<td>Endpoint</td>
<td>String</td>
<td>The endpoint to check for subscription to the specified topic. The endpoint can be an email address, an Amazon SQS queue, or any other endpoint type supported by Amazon SNS.</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
<tr>
<td>True</td>
<td>Any</td>
<td>The ARN is subscribed to the Amazon SNS topic</td>
</tr>
<tr>
<td>False</td>
<td>Any</td>
<td>The ARN is not subscribed to the Amazon SNS topic</td>
</tr>
</tbody>
</table>

### SubscribeToTopic node

Subscribes to an Amazon SNS topic.

| Cloud Canvas (AWS):AWS Services:SNS (Notification service):SubscribeToTopic |
|-----------------------------|-----------------------------|-----------------------------|
| Subscribe                  | Success                     |
| Protocol                   | Error                       |
| TopicARN                   | SubscriptionArn             |

### Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subscribe</td>
<td>Any</td>
<td>Subscribes to a topic to receive messages published to that topic. For more information, see Subscribe to a Topic.</td>
</tr>
<tr>
<td>Protocol</td>
<td>String</td>
<td>The protocol of the endpoint to which to subscribe</td>
</tr>
<tr>
<td>TopicARN</td>
<td>String</td>
<td>The ARN of the Amazon SNS topic to which to subscribe</td>
</tr>
<tr>
<td>Endpoint</td>
<td>String</td>
<td>The address of the endpoint to subscribe (for example, an email address). For information on sending to HTTP or HTTPS, see Sending Amazon SNS Messages to HTTP/HTTPS Endpoints.</td>
</tr>
</tbody>
</table>
SQS (Message Queuing Service) Nodes

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 about the latest features, see the Amazon Lumberyard User Guide.

For more information on Amazon SNS, see the Amazon Simple Notification Service Developer Guide.

You can use these flow graph nodes to start polling AWS queues and to push messages to AWS queues.

Topics
- PollAndNotify node (p. 713)
- Push node (p. 714)

PollAndNotify node

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Any</td>
<td>Start polling an AWS queue</td>
</tr>
<tr>
<td>QueueName</td>
<td>String</td>
<td>The name of an AWS queue that has already been created</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
<tr>
<td>OnMessageReceived</td>
<td>String</td>
<td>The most recent message on the stack</td>
</tr>
<tr>
<td>QueueArn</td>
<td>String</td>
<td>The ARN (Amazon Resource Name) of the queue</td>
</tr>
</tbody>
</table>
Push node

Pushes a message to an AWS queue

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Push</td>
<td>Any</td>
<td>Pushes a message to an AWS queue</td>
</tr>
<tr>
<td>QueueName</td>
<td>String</td>
<td>The name of an AWS queue that has already been created</td>
</tr>
<tr>
<td>Message</td>
<td>String</td>
<td>The message to send</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
</tbody>
</table>

Static Data (PROTOTYPE) Nodes

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 about the latest features, see the Amazon Lumberyard User Guide.

Static Data is a Lumberyard system for managing game data that changes less frequently through monitored Amazon S3 buckets. You can use these flow graph nodes to update or query your buckets and/or monitor them at regular intervals for changes.

Topics
- Add Monitored Bucket node (p. 714)
- Get Static Data node (p. 715)
- Load Static Data node (p. 716)
- Remove Monitored Bucket node (p. 716)
- Request Bucket node (p. 717)
- Set Update Frequency node (p. 717)

Add Monitored Bucket node

Adds an Amazon S3 bucket to monitor.
**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddBucket</td>
<td>Void</td>
<td>Adds a bucket to watch for updates</td>
</tr>
<tr>
<td>BucketName</td>
<td>String</td>
<td>The name of the Amazon S3 bucket to watch</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
<tr>
<td>Finished</td>
<td>String</td>
<td>The bucket was added</td>
</tr>
</tbody>
</table>

**Get Static Data node**

Retrieves a field from a static data definition.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get</td>
<td>Void</td>
<td>Retrieves a value from static data</td>
</tr>
<tr>
<td>StaticDataType</td>
<td>String</td>
<td>The type of the static data to retrieve</td>
</tr>
<tr>
<td>StaticDataId</td>
<td>String</td>
<td>The identifier for the static data definition in the table</td>
</tr>
<tr>
<td>StaticDataField</td>
<td>String</td>
<td>The field name of the data to retrieve</td>
</tr>
<tr>
<td>ActivateOnUpdate</td>
<td>Void</td>
<td>Fire the node again the next time an update of the data takes place</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
</tbody>
</table>
### Load Static Data node

Attempts to load static data of the type specified.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>Any</td>
<td>Load a type of static data</td>
</tr>
<tr>
<td>StaticDataType</td>
<td>String</td>
<td>The type of static data to load</td>
</tr>
</tbody>
</table>

**Outputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
<tr>
<td>Finished</td>
<td>String</td>
<td>Finished attempting to load</td>
</tr>
</tbody>
</table>

### Remove Monitored Bucket node

Removes a bucket name from the list of monitored buckets.

**Inputs**

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remove</td>
<td>Any</td>
<td>Removes a bucket from the list of monitored buckets</td>
</tr>
<tr>
<td>BucketName</td>
<td>String</td>
<td>The name of the bucket to remove</td>
</tr>
</tbody>
</table>
Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
<tr>
<td>Finished</td>
<td>String</td>
<td>Finished removing the bucket</td>
</tr>
</tbody>
</table>

Request Bucket node

Requests an update of a specific bucket, or of all monitored buckets.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RequestBucket</td>
<td>Any</td>
<td>Requests an update of a specific bucket or of all monitored buckets</td>
</tr>
<tr>
<td>BucketName</td>
<td>String</td>
<td>The name of the bucket for which to request an update. To request updates for all buckets, leave this value blank.</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
<tr>
<td>Finished</td>
<td>String</td>
<td>Finished sending the request</td>
</tr>
</tbody>
</table>

Set Update Frequency node

Sets or clears a recurring timer to poll monitored buckets.

Inputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetTimer</td>
<td>Void</td>
<td>Sets a recurring timer to the value specified in TimerValue</td>
</tr>
</tbody>
</table>
### Static Data (PROTOTYPE) Nodes

#### Port

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimerValue</td>
<td>Integer</td>
<td>The time interval at which to poll. Possible values are from 0 to 100. A value of 0 clears the timer; 0 is the default.</td>
</tr>
</tbody>
</table>

#### Outputs

<table>
<thead>
<tr>
<th>Port</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success</td>
<td>Any</td>
<td>Activated upon a successful operation</td>
</tr>
<tr>
<td>Error</td>
<td>String</td>
<td>Activated upon an error being detected; the value of the port is the error message</td>
</tr>
<tr>
<td>Set</td>
<td>String</td>
<td>The timer has been set</td>
</tr>
</tbody>
</table>
Input in Amazon Lumberyard

This section contains information on legacy game input in Amazon Lumberyard.

For information on Lumberyard's new input system, see Input in Amazon Lumberyard

Topics

• Action Maps (p. 719)
• CryInput (p. 719)
• Setting Up Controls and Action Maps (p. 721)

Action Maps

Note

Lumberyard’s Input component replaces legacy action maps. For more information, see Input in the Amazon Lumberyard User Guide.

The Action Map Manager provides a high-level interface to handle input controls inside a game. The Action Map system is implemented in Lumberyard, and can be used directly by any code inside Lumberyard or the GameDLL.

Initializing the Action Map Manager

The Action Map Manager is initialized when Lumberyard is initialized. Your game must specify the path for the file defaultProfile.xml (by default, the path is Game/Libs/Config/defaultProfile.xml). You can do this by passing the path to the manager. For example:

```cpp
IActionMapManager* pActionMapManager = m_pFramework->GetIActionMapManager();
if (pActionMapManager)
{
    pActionMapManager->InitActionMaps(filename);
}
```

Upon initialization, the Action Map Manager clears all existing initialized maps, filters, and controller layouts.

Receiving Actions During Runtime

You can enable the feature that allows action maps to receive actions during runtime. Use the following code to enable or disable an action map during runtime:

```cpp
pActionMapMan->EnableActionMap("default", true);
```

To receive actions, implement the IActionListener interface in a class.

CryInput

The main purpose of CryInput is to provide an abstraction that obtains input and status from various input devices such as a keyboard, mouse, joystick, and so on.
It also supports sending feedback events back to input devices—for example, in the form of force feedback events.

The common interfaces for the input system can be found in IInput.h, in the CryCommon project.

### IInput

IInput is the main interface of the input system. An instance implementing this interface is created automatically during system initialization in the InitInput function (InitSystem.cpp in CrySystem, see also CryInput.cpp in CryInput).

Only one instance of this interface is created. CrySystem also manages the update and shutdown of the input system.

This IInput instance is stored in the SSysGlobalEnvironment structure gEnv. You can access it through gEnv->pInput or, alternatively, through the system interface by GetISystem()->GetIInput(). Access through the gEnv variable is the most commonly used method.

### IInputEventListener

A common use case within the input system is to create listener classes in other modules (for example, CryGame) by inheriting from IInputEventListener and registering/unregistering the listener class with the input system for notifications of input events.

For example, the Action Map System registers itself as an input listener and forwards game events only for the keys defined in the profile configuration files to further abstract the player input from device to the game.

### SInputEvent

SInputEvent encapsulates information that is created by any input device and received by all input event listeners.

### IInputDevice

Input devices normally relate directly to physical input devices such as a joystick, mouse, keyboard, and so on. To create a new input device, you must implement all functions in the IInputDevice interface and register an instance of it with the Input System using the AddInputDevice function.

The Init function is called when registering the IInputDevice with the Input System; it is not necessary to manually call it when creating the input devices.

The Update function is called at every update of the Input System—this is generally where the state of the device should be checked/updated and the Input Events generated and forwarded to the Input System.

It is common for input devices to create and store a list in SInputSymbol of each symbol the input device is able to generate in the Init function. Then, in the update function, the symbols for the buttons/axes that changed are looked up and used (via their AssignTo function) to fill in most of the information needed for the events, which are then forwarded to the input system.

**Example:**

```cpp
// function from CInputDevice (accessible only within CryInput)
MapSymbol(...)```

720
Setting Up Controls and Action Maps

This section describes how to create and modify action maps to customize the controls to the needs of your game.

Action map profiles for all supported operating systems and devices are located in `Game\Libs\Config\Profile\DefaultProfile.xml`. This default XML file organizes controls into the following sections, each of which is controlled by its own action map:

- multiplayer
- singleplayer

To forward events to the input system so that event listeners can receive them, use the `PostInputEvent` function from `IInput`.

If adding your input device to CryInput, it may be useful to inherit directly from `CInputDevice`, as it already provides a generic implementation for most functions in `IInputDevice`.

**Note**
This file is included with the full source of CryEngine and is not available in the FreeSDK or GameCodeOnly solutions. For these licenses please derive from `IInputDevice` directly.
• debug
• flycam
• default
• player
• vehicle
• land vehicle
• sea vehicle
• helicopter

Each action map can be enabled or disabled during runtime from Flow Graph, in Lua scripts, or in C++ code.

See the topic Default Controller Mapping (p. 724) for an overview of the controls in the SDK package.

**Action Maps**

An action map is a set of key/button mappings for a particular game mode. For example, there is an `<actionmap>` section for helicopter controls called "Helicopter", which means that everything inside that section consists of key and button bindings that apply only when flying a helicopter. To change your common in-game bindings, go to the section starting with `<actionmap name="default">`. There are also sections for multiplayer-specific bindings and, of course, any other vehicles or modes you need.

The following is an overview of a standard action map, in this case the standard debug one:

```xml
<actionmap name="debug" version="22">
<!-- debug keys – move to debug when we can switch devmode-->
<action name="flymode" onPress="1" noModifiers="1" keyboard="f3" />
<action name="godmode" onPress="1" noModifiers="1" keyboard="f4" />
<action name="toggleaidebugdraw" onPress="1" noModifiers="1" keyboard="f11" />
<action name="togglepdrawhelpers" onPress="1" noModifiers="1" keyboard="f10" />
<action name="ulammo" onPress="1" noModifiers="1" keyboard="np_2" />
<action name="debug" onPress="1" keyboard="7" />
<action name="thirdperson" onPress="1" noModifiers="1" keyboard="f1" />
<!-- debug keys – end -->
</actionmap>
```

**Versioning**

```xml
<actionmap name="debug" version="22">
```

When the version value is incremented, Lumberyard ensures that the user profile receives the newly updated action map. This is quite useful when deploying new actions in a patch of a game that is already released. If the version stays the same, changes or additions to the action maps are not propagated to the user profile.

**Activation Modes**

The following activation modes are available:

• onPress – The action key is pressed
• onRelease – The action key is released
• onHold – The action key is held
• always – Permanently activated
The activation mode is passed to action listeners and identified by the corresponding Lua constant:

- eAAM_OnPress
- eAAM_OnRelease
- eAAM_OnHold
- eAAM_Always

Modifiers available:

- retriggerable
- holdTriggerDelay
- holdRepeatDelay
- noModifiers – Action takes place only if no Ctrl, Shift, Alt, or Win keys are pressed
- consoleCmd – Action corresponds to a console command
- pressDelayPriority
- pressTriggerDelay
- pressTriggerDelayRepeatOverride
- inputsToBlock – Specify the input actions to block here
- inputBlockTime – Time to block the specified input action

**Action Filters**

You can also define action filters directly in your defaultProfile.xml file. The following attributes are available:

- **name** – How the filter will be identified.
- **type** – Specify actionFail to cause an action to fail. Specify actionPass to allow the action to succeed.

A sample action filter follows:

```xml
<actionfilter name="no_move" type="actionFail">
  <!-- actions that should be filtered -->
  <action name="crouch"/>
  <action name="jump"/>
  <action name="moveleft"/>
  <action name="moveright"/>
  <action name="moveforward"/>
  <action name="moveback"/>
  <action name="sprint"/>
  <action name="xi_movey"/>
  <action name="xi_movex"/>
  <!-- actions end -->
</actionfilter>
```

**Controller Layouts**

Links to the different controller layouts can also be stored in this file:

```xml
<controllerlayouts>
  <layout name="Layout 1" file="buttonlayout_alt.xml"/>
  <layout name="Layout 2" file="buttonlayout_alt2.xml"/>
</controllerlayouts>
```
Working with Action Maps During Runtime

In Lumberyard, you can use the console command `i_reloadActionMaps` to re-initialize the defined values. The ActionMapManager sends an event to all its listeners to synchronize the values throughout the engine. If you're using a separate GameActions file like GameSDK, make sure this class will receive the update to re-initialize the actions/filters in place. Keep in mind that it's not possible to define action maps, filters, or controller layouts with the same name in multiple places (for example, action filter `no_move` defined in `defaultProfile.xml` and the `GameActions` file).

To handle actions during runtime, you can use flow graphs.

- Flow Graph – Input nodes can be used to handle actions. Only digital inputs can be handled from a flow graph.

Default Controller Mapping

The default mapping for input on the PC is shown in the following table. To reconfigure the controls for your game, follow the instructions in Setting Up Controls and Action Maps (p. 721) and Action Maps (p. 719).

<table>
<thead>
<tr>
<th>Player Action</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Player Movement</td>
<td>W, A, S, D</td>
</tr>
<tr>
<td>Player Aim</td>
<td>Mouse XY</td>
</tr>
<tr>
<td>Jump</td>
<td>Spacebar</td>
</tr>
<tr>
<td>Sprint</td>
<td>Shift</td>
</tr>
<tr>
<td>Crouch</td>
<td>C</td>
</tr>
<tr>
<td>Slide (when sprinting)</td>
<td>C</td>
</tr>
<tr>
<td>Fire</td>
<td>Mouse 1</td>
</tr>
<tr>
<td>Zoom</td>
<td>Mouse 2</td>
</tr>
<tr>
<td>Melee</td>
<td>V</td>
</tr>
<tr>
<td>Fire Mode</td>
<td>2</td>
</tr>
<tr>
<td>Reload</td>
<td>R</td>
</tr>
<tr>
<td>Use</td>
<td>F</td>
</tr>
<tr>
<td>Toggle Weapon</td>
<td>1</td>
</tr>
<tr>
<td>Toggle Explosive</td>
<td>3</td>
</tr>
<tr>
<td>Toggle Binoculars</td>
<td>B</td>
</tr>
</tbody>
</table>
### Key Naming Conventions

This page lists some of the name conventions used for action maps.

<table>
<thead>
<tr>
<th>Player Action</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toggle Light (attachment)</td>
<td>L</td>
</tr>
<tr>
<td>Third Person Camera</td>
<td>F1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vehicle Action</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerate</td>
<td>W</td>
</tr>
<tr>
<td>Boost</td>
<td>Shift</td>
</tr>
<tr>
<td>Brake/Reverse</td>
<td>S</td>
</tr>
<tr>
<td>Handbrake</td>
<td>Spacebar</td>
</tr>
<tr>
<td>Steer</td>
<td>A/D</td>
</tr>
<tr>
<td>Look</td>
<td>Mouse XY</td>
</tr>
<tr>
<td>Horn</td>
<td>H</td>
</tr>
<tr>
<td>Fire</td>
<td>Mouse 1</td>
</tr>
<tr>
<td>Change Seat</td>
<td>C</td>
</tr>
<tr>
<td>Headlights</td>
<td>L</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Helicopter Action</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ascend</td>
<td>W</td>
</tr>
<tr>
<td>Descend</td>
<td>S</td>
</tr>
<tr>
<td>Roll Left</td>
<td>A</td>
</tr>
<tr>
<td>Roll Right</td>
<td>D</td>
</tr>
<tr>
<td>Yaw Left</td>
<td>Mouse X (left)</td>
</tr>
<tr>
<td>Yaw Right</td>
<td>Mouse X (right)</td>
</tr>
<tr>
<td>Pitch Up</td>
<td>Mouse Y (up)</td>
</tr>
<tr>
<td>Pitch Down</td>
<td>Mouse Y (down)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multiplayer Action</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Scoreboard</td>
<td>TAB</td>
</tr>
</tbody>
</table>
### Key Gestures

<table>
<thead>
<tr>
<th>Letters</th>
<th>&quot;a&quot; - &quot;z&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numbers</td>
<td>&quot;1&quot; - &quot;0&quot;</td>
</tr>
<tr>
<td>Arrows</td>
<td>&quot;up&quot;, &quot;down&quot;, &quot;left&quot;, &quot;right&quot;</td>
</tr>
<tr>
<td>Function keys</td>
<td>&quot;f1&quot; - &quot;f15&quot;</td>
</tr>
<tr>
<td>Numpad</td>
<td>&quot;np_1&quot; - &quot;np_0&quot;, &quot;numlock&quot;, &quot;np_divide&quot;, &quot;np_multiply&quot;, &quot;np_subtract&quot;, &quot;np_add&quot;, &quot;np_enter&quot;, &quot;np_period&quot;</td>
</tr>
<tr>
<td>Esc</td>
<td>&quot;escape&quot;</td>
</tr>
<tr>
<td>~</td>
<td>&quot;tilde&quot;</td>
</tr>
<tr>
<td>Tab</td>
<td>&quot;tab&quot;</td>
</tr>
<tr>
<td>CapsLock</td>
<td>&quot;capslock&quot;</td>
</tr>
<tr>
<td>Shift</td>
<td>&quot;lshift&quot;, &quot;rshift&quot;</td>
</tr>
<tr>
<td>Ctrl</td>
<td>&quot;lctrl&quot;, &quot;rctrl&quot;</td>
</tr>
<tr>
<td>Alt</td>
<td>&quot;lalt&quot;, &quot;ralt&quot;</td>
</tr>
<tr>
<td>spacebar</td>
<td>&quot;space&quot;</td>
</tr>
<tr>
<td>-</td>
<td>&quot;minus&quot;</td>
</tr>
<tr>
<td>=</td>
<td>&quot;equals&quot;</td>
</tr>
<tr>
<td>Backspace</td>
<td>&quot;backspace&quot;</td>
</tr>
<tr>
<td>[ ]</td>
<td>&quot;lbracket&quot;, &quot;rbracket&quot;</td>
</tr>
<tr>
<td>&quot;&quot;</td>
<td>&quot;backslash&quot;</td>
</tr>
<tr>
<td>;</td>
<td>&quot;semicolon&quot;</td>
</tr>
<tr>
<td>'</td>
<td>&quot;apostrophe&quot;</td>
</tr>
<tr>
<td>Enter</td>
<td>&quot;enter&quot;</td>
</tr>
<tr>
<td>,</td>
<td>&quot;comma&quot;</td>
</tr>
<tr>
<td>.</td>
<td>&quot;period&quot;</td>
</tr>
<tr>
<td>/</td>
<td>&quot;slash&quot;</td>
</tr>
<tr>
<td>Home</td>
<td>&quot;home&quot;</td>
</tr>
<tr>
<td>End</td>
<td>&quot;end&quot;</td>
</tr>
<tr>
<td>Delete</td>
<td>&quot;delete&quot;</td>
</tr>
<tr>
<td>PageUp</td>
<td>&quot;pgup&quot;</td>
</tr>
<tr>
<td>PageDown</td>
<td>&quot;pgdn&quot;</td>
</tr>
<tr>
<td>Insert</td>
<td>&quot;insert&quot;</td>
</tr>
<tr>
<td>ScrollLock</td>
<td>&quot;scrolllock&quot;</td>
</tr>
</tbody>
</table>
### Key Naming Conventions

<table>
<thead>
<tr>
<th>Action</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrintScreen</td>
<td>&quot;print&quot;</td>
</tr>
<tr>
<td>Pause/Break</td>
<td>&quot;pause&quot;</td>
</tr>
</tbody>
</table>

### Mouse Gestures

<table>
<thead>
<tr>
<th>Gesture</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left/primary mouse button</td>
<td>&quot;mouse1&quot;</td>
</tr>
<tr>
<td>Right/secondary mouse button</td>
<td>&quot;mouse2&quot;</td>
</tr>
<tr>
<td>Mouse wheel up</td>
<td>&quot;mwheel_up&quot;</td>
</tr>
<tr>
<td>Mouse wheel down</td>
<td>&quot;mwheel_down&quot;</td>
</tr>
<tr>
<td>New position along x-axis</td>
<td>&quot;maxis_x&quot;</td>
</tr>
<tr>
<td>New position along y-axis</td>
<td>&quot;maxis_y&quot;</td>
</tr>
</tbody>
</table>
Lua Scripting Reference

CryLua is deprecated and will be removed in a future version of Lumberyard.

The topics in this section document Lua functionality for the legacy script context.

Starting with Lumberyard 1.8, Lua scripts use the new behavior context that replaces the legacy script context.

Scripts that were written before the integration of the behavior context no longer work in Lumberyard versions 1.8 and later. For information on updating Lua code from legacy script context to the new behavior context, see the migration notes for Lumberyard 1.8.

For information on using Lua with Lumberyard’s new component entity system, see Writing Lua Scripts for the Component Entity System.

Topics
- Working with Lua Scripting (p. 728)
- Loading Canvases in Lua (p. 734)
- Entity System Script Callbacks (p. 735)
- Game Rules Script Callbacks (p. 737)
- Common Lua Globals and Functions (p. 739)
- EntityUtils Lua Functions (p. 743)
- Lua Vector and Math Functions (p. 746)
- Physics Lua Functions (p. 756)
- Lua ScriptBind Reference (p. 757)
- Integrating Lua and C++ (p. 964)

Working with Lua Scripting

CryLua is deprecated and will be removed in a future version of Lumberyard.

Lumberyard uses Lua for its scripting language.

The Entity system can attach a script proxy to any entity, which is in the form of a table that can include data and functions. AI behaviors are often written in scripts. Additionally, several game systems, including Actor, Item, Vehicle, and GameRules, rely on scripting to extend their functionality.

The advantages of using scripts include:

- Fast iteration – Scripts can be reloaded within the engine.
- Runtime performance – Careful usage of available resources can result into scripts that run nearly as fast as compiled code.
Running Scripts

You can run scripts either by calling script files directly from code or by using console commands.

In code

Scripts are stored in the \Game\Scripts directory. To invoke a script file, call the LoadScript function from your C++ code. For more information, see Integrating Lua and C++ (p. 964). Another option is to create a script entity, as described in Entity Scripting (p. 1050).

In the Console

Script instructions can be executed using the in-game console. This can be done by appending the # character before the instructions. This functionality is limited to Lumberyard Editor or when running the launcher in dev mode (using the -DEVMODE command-line argument).

Reloading Scripts During Runtime

In Lumberyard Editor it is always possible to reload entities within the user interface. When reloading a script entity, choose the Reload Script button, which is found in the Rollup Bar.

You can also use the following ScriptBind functions to reload scripts.

- Script.ReloadScript(filename)
- Script.ReloadScripts()

To invoke these functions from the console, use the following syntax:

```
#Script.ReloadScript("Scripts\EntityCommon.lua")
```

Recommended Reading

The following resources on the Lua language are recommended reading when working with scripts with Lumberyard.

- Lua 5.1 Reference Manual
- Programming in Lua, Third Edition
- Other books

Topics

- Using the Lua Remote Debugger (p. 729)
- Using the Lua XML Loader (p. 731)

Using the Lua Remote Debugger

Lumberyard includes a standalone visual script debugger for Lua. To start the debugger, you first enable it in the console, and then run the LuaRemoteDebugger.exe executable file.
1. In Lumberyard Editor console or the game console, type `lua_debugger 1` or `lua_debugger 2`. This enables enable debugging in one of the following two modes:
   - Mode 1 – The debugger breaks on both breakpoints and script errors.
   - Mode 2 – The debugger breaks only on script errors.
2. Run the Lua remote debugger executable file at the Lumberyard directory location `\dev\Tools\LuaRemoteDebugger\LuaRemoteDebugger.exe`.
3. In the Lua remote debugger, on the **File** menu, choose **Connect**.
4. If you are running the game in the editor (you pressed Ctrl+G) and want to debug your scripts, choose **PC (Editor)**. If you want to attach the debugger to the built game executable, choose **PC (Game)**.
   - For **IP address** and **Port**, type the IP address and port of the computer to which you want to connect. The default options connect to the game on your local computer. The default IP address is 127.0.01 (localhost). For **PC (Editor)**, the default port is 9433. For **PC (Game)**, the default port is 9432.
5. Choose **Connect**. In Lumberyard Editor, the console window displays **Lua remote debug client connected**.

The first time you run Lua remote debugger, it prompts you for the scripts folder:

```
Found scripts folder at C:\\dev\\Cache\\samplesproject\\samplesproject\\Scripts. Would you like to set this as your scripts folder?
```

The default folder is the Scripts folder of the project that you are running. For example, if you are running the samples project, the folder is `samplesproject/Scripts`.

6. To accept the default location, click **Yes**.
   - **Note**
   - To change the scripts folder location, choose **File, Set Scripts Folder**.

   After you choose the location for your scripts folder, the folder's contents are shown in the navigation tree on the left.

### Performing Tasks in the Lua Remote Debugger

To perform specific tasks in the Lua remote debugger, see the following table:

<table>
<thead>
<tr>
<th>To do this</th>
<th>Do this</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open a script file</td>
<td>Double click the script file in the navigation pane, or press Ctrl+O to open the <strong>Find File</strong> dialog.</td>
</tr>
<tr>
<td>Set a break point</td>
<td>Place the cursor on the line in the script where you want the break to occur, and then click the red dot in the toolbar or press F9. When program execution stops on a break point, the <strong>Call Stack</strong> and <strong>Locals</strong> tabs populate.</td>
</tr>
<tr>
<td>Remove a break point</td>
<td>Place the cursor on the line with the breakpoint that you want to remove, and then click the red dot in the toolbar or press F9.</td>
</tr>
</tbody>
</table>
Using the Lua XML Loader

CryLua is deprecated and will be removed in a future version of Lumberyard.

There is a generic interface for parsing and translating XML files into Lua files. This interface uses an XML file as a definition format that declares what kind of XML is included in a file and what kind of Lua to create from the XML. The format includes some simple validation methods to ensure that the data received is what is expected.

**XML Data**

The XML loader can distinguish between three kinds of data: properties, arrays, and tables.

**Tables**

This table represents a Lua-based table:

```lua
letters = { a="a", b="b", c="c" }
```
In an XML data file, this table would look like this:

```
<letters a="a" b="b" c="c"/>
```

The XML definition file would look like this:

```
<Table name="letters">
    <Property name="a" type="string"/>
    <Property name="b" type="string"/>
    <Property name="c" type="string"/>
</Table>
```

Each element can be marked as optional in the definition file using the attribute `optional="1"`.

**Arrays**

There are two possible types of arrays. The first type is a simple group of elements, shown in Lua like this:

```
numbers = {0,1,2,3,4,5,6,7,8,9}
```

In the XML data file, the array would look like this:

```
<numbers>
    <number value="0"/>
    <number value="1"/>
    <number value="2"/>
    <number value="3"/>
    <number value="4"/>
    <number value="5"/>
    <number value="6"/>
    <number value="7"/>
    <number value="8"/>
    <number value="9"/>
</numbers>
```

The data definition file would look like this:

```
<Array name="numbers" type="int" elementName="number"/>
```

The second array type is an array of tables. In Lua:

```
wheels = {
    {size=3, weight=10},
    {size=2, weight=1},
    {size=4, weight=20},
}
```

In the XML data file:

```
<wheels>
    <wheel size="3" weight="10"/>
    <wheel size="2" weight="1"/>
    <wheel size="4" weight="20"/>
</wheels>
```
The XML definition file:

```xml
<Array name="wheels" elementName="wheel"> <!-- note no type is attached -->
    <Property name="size" type="float"/>
    <Property name="weight" type="int"/>
</Array>
```

**Loading and Saving a Table from Lua**

To load and initialize a Lua table:

```lua
someTable = CryAction.LoadXML( definitionFileName, dataFileName )
```

When storing XML files for scripts, the recommended practice is to keep the definition files with the scripts that use them, but store the data files in a directory outside the Scripts directory.

To save a table from Lua:

```lua
CryAction.SaveXML( definitionFileName, dataFileName, table )
```

**Data Types**

The following data types are available, and can be set wherever a "type" attribute is present in the definition file.

- float – Floating point number.
- int – Integer.
- string – String.
- bool – Boolean value.
- Vec3 – Floating point vectors with three components. Values of this type are expressed as follows:
  - XML – "1,2,3"
  - Lua – {x=1,y=2,z=3}

**Enums**

For string type properties, an optional `<Enum>` definition can be used. Property values will be validated against the enum.

Example:

```xml
<Property name="view" type="string">
    <Enum>
        <Value>GhostView</Value>
        <Value>ThirdPerson</Value>
        <Value>BlackScreen</Value>
    </Enum>
</Property>
```

Enum support for other data types can be added, if necessary.

**Example**

XML definition file:
Loading Canvases in Lua

CryLua is deprecated and will be removed in a future version of Lumberyard.

The Lumberyard UI Editor uses the concept of a UI canvas as an invisible backdrop for user interface elements. You can use the Lua scripting language to load and unload UI canvases in Lumberyard.

To load a canvas in Lua

1. Create a new, plain text file in your game project directory with a .lua file extension.
2. Type or paste the following sample script into your new Lua file:

   ```lua
   loadcanvas = {
   Properties =
   }
   ```

   **Note**
   The following script uses a Lua file named loadcanvas.lua and loads a canvas file named menu.uicanvas saved at the root of the game project directory. Substitute the appropriate file names for your script.
function loadcanvas:OnActivate()
    self.uiCanvasLuaProxy = UiCanvasLuaProxy()
    self.uiCanvasLuaProxy:LoadCanvas("menu.uicanvas")
end

3. In Lumberyard Editor, right-click in the Viewport and click Create Component Entity.

4. If the Entity Inspector does not open automatically, click Tools, Entity Inspector.

5. Click Add Component.

6. Select Scripting, Lua Script.

7. Under Lua Script, click ... and open the Lua script file that you created.

8. In Lumberyard Editor, click Game, Switch to Game to enter game mode. Verify that your canvas file loads.

See Also

For more information, consult the following resources.

- For a full listing of UI Lua API calls, see UI Lua Reference.
- For general information about UI canvases, see Working with UI Canvases in the Amazon Lumberyard User Guide.

Entity System Script Callbacks

CryLua is deprecated and will be removed in a future version of Lumberyard.

This topic describes all callbacks for the Entity system. Use of these callbacks functions is not obligatory, but some cases require that entities behave properly within Lumberyard Editor. For example, the OnReset callback should be used to clean the state when a user enters or leaves the game mode within Lumberyard Editor.

Default State Functions

<table>
<thead>
<tr>
<th>Callback Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnSpawn</td>
<td>Called after an entity is created by the Entity system.</td>
</tr>
<tr>
<td>OnDestroy</td>
<td>Called when an entity is destroyed (like OnShutDown() gets called).</td>
</tr>
<tr>
<td>OnInit</td>
<td>Called when an entity gets initialized via ENTITY_EVENT_INIT, and when its ScriptProxy gets initialized.</td>
</tr>
<tr>
<td>OnShutDown</td>
<td>Called when an entity is destroyed (like OnDestroy() gets called).</td>
</tr>
<tr>
<td>OnReset</td>
<td>Usually called when an editor wants to reset the state.</td>
</tr>
<tr>
<td>OnPropertyChange</td>
<td>Called by Lumberyard Editor when the user changes one of the properties.</td>
</tr>
</tbody>
</table>
## Script State Functions

<table>
<thead>
<tr>
<th>Callback Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OnBeginState</strong></td>
<td>Called during <code>Entity.GotoState()</code> after the state has been changed (that is, after <code>OnEndState()</code> is called on the old state).</td>
</tr>
<tr>
<td><strong>OnBind</strong></td>
<td>Called when a child entity is attached to an entity. Parameters include:</td>
</tr>
<tr>
<td></td>
<td>• script table for the child entity</td>
</tr>
<tr>
<td><strong>OnCollision</strong></td>
<td>Called when a collision between an entity and something else occurs. Parameters include:</td>
</tr>
<tr>
<td></td>
<td>• script table with information about the collision</td>
</tr>
<tr>
<td><strong>OnEndState</strong></td>
<td>Called during <code>Entity.GotoState()</code> while the old state is still active and before <code>OnBeginState()</code> is called on the new state.</td>
</tr>
<tr>
<td><strong>OnEnterArea</strong></td>
<td>Called when an entity has fully entered an area or trigger. Parameters include:</td>
</tr>
<tr>
<td></td>
<td>• areaId (int)</td>
</tr>
<tr>
<td></td>
<td>• fade_fraction (float) This value is 1.0f if the entity has fully entered the area, or 0.0f in the case of trigger boxes.</td>
</tr>
<tr>
<td><strong>OnEnterNearArea</strong></td>
<td>Called when an entity enters the range of an area. Works with Box-, Sphere- and Shape-Areas if a sound volume entity is connected. Takes OuterRadius of sound entity into account to determine when an entity is near the area.</td>
</tr>
<tr>
<td><strong>OnLeaveArea</strong></td>
<td>Called when an entity has fully left an area or trigger. Parameters include:</td>
</tr>
<tr>
<td></td>
<td>• areaId (int)</td>
</tr>
<tr>
<td></td>
<td>• fade_fraction (float) This value is always 0.0f.</td>
</tr>
<tr>
<td><strong>OnLeaveNearArea</strong></td>
<td>Called when an entity leaves the range of an area. Works with Box-, Sphere- and Shape-Areas if a sound volume entity is connected. Takes OuterRadius of sound entity into account to determine when an entity is near the area.</td>
</tr>
<tr>
<td><strong>OnMove</strong></td>
<td>Called whenever an entity moves through the world.</td>
</tr>
<tr>
<td><strong>OnMoveNearArea</strong></td>
<td>Called when an entity moves. Works with Box-, Sphere- and Shape-Areas if a sound volume entity is connected. Takes OuterRadius of sound entity into account to determine when an entity is near the area.</td>
</tr>
<tr>
<td><strong>OnProceedFadeArea</strong></td>
<td>Called when an entity has recently entered an area and fading is still in progress. Parameters include:</td>
</tr>
<tr>
<td></td>
<td>• areaId (int)</td>
</tr>
<tr>
<td></td>
<td>• fade_fraction (float)</td>
</tr>
<tr>
<td><strong>OnSoundDone</strong></td>
<td>Called when a sound stops. Parameters include:</td>
</tr>
<tr>
<td></td>
<td>• soundId (int) The ID of the sound played, which was provided with the request to play the sound.</td>
</tr>
<tr>
<td><strong>OnStartGame</strong></td>
<td>Called when a game is started.</td>
</tr>
<tr>
<td><strong>OnStartLevel</strong></td>
<td>Called when a new level is started.</td>
</tr>
</tbody>
</table>
### Callback Function Description

**OnTimer**
Called when a timer expires. Parameters include:
- **timerId** (int) The ID of the time, provided by `Entity.SetTimer()`.
- **period** (int) Length of time, in milliseconds, that the timer runs

**OnUnBind**
Called when a child entity is about to be detached from an entity. Parameters include:
- script table for the child entity

**OnUpdate**
Called periodically by the engine on the entity's current state. This assumes the console variable `es_UpdateScript` is set to 1.

### Game Rules Script Callbacks

CryLua is deprecated and will be removed in a future version of Lumberyard.

This topic provides reference information on callbacks used with the GameRules scripts.

<table>
<thead>
<tr>
<th>Callback Function</th>
<th>Description</th>
</tr>
</thead>
</table>
| **OnAddTaggedEntity** | Called when a player is added as a tagged player on the minimap. Called on the server only.  
- **shooterId** – Entity that tagged the target player.  
- **targetId** – Tagged player. |
| **OnClientConnect** | Called when a player connects. Called on the server only.  
- **channelId** |
| **OnClientDisconnect** | Called when a player disconnects. Called on the server only.  
- **channelId** |
| **OnClientEnteredGame** | Called when a player enters the game and is part of the game world. Called on the server only.  
- **channelId** – Channel identifier of the player.  
- **playerScriptTable** – The player's script table.  
- **bReset** – Boolean indicating whether or not the channel is from the reset list.  
- **bLoadingSaveGame** – Boolean indicating whether or not the call was made during a saved game loading. |
| **OnDisconnect** | Called when the player disconnects on the client. Called on the client only.  
- **cause** – Integer identifying the disconnection cause. See `EDisconnectionCause`.  
- **description** – Human readable description of the disconnection cause. |
| **OnChangeSpectatorMode** | Called when a player changes the spectator mode. Called on the server only. |
### Callback Function

**Description**

- **entityId** – Player who made the change.
- **mode** – New spectator mode (1=fixed, 2=free, 3= follow).
- **targetId** – Possible target entity to spectate.
- **resetAll** – Boolean indicating whether or not to reset player-related things like the inventory.

#### OnChangeTeam

Called when a player switches teams. Called on the server only.

- **entityId** – Player who switched teams.
- **teamId** – New team identifier.

#### OnExplosion

Called when an explosion is simulated. Called on the server and client.

- **pos** – Position of the explosion in the game world.
- **dir** – Direction of the explosion.
- **shooterId**
- **weaponId**
- **shooter**
- **weapon**
- **materialId**
- **damage**
- **min_radius**
- **radius**
- **pressure**
- **hole_size**
- **effect**
- **effectScale**
- **effectClass**
- **typeId**
- **type**
- **angle**
- **impact**
- **impact_velocity**
- **impact_normal**
- **impact_targetId**
- **shakeMinR**
- **shakeMaxR**
- **shakeScale**
- **shakeRnd**
- **impact**
- **impact_velocity**
- **impact_normal**
- **impact_targetId**
- **AffectedEntities** – Affected entities table.
- **AffectedEntitiesObstruction** – Affected entities obstruction table.
Common Lua Globals and Functions

CryLua is deprecated and will be removed in a future version of Lumberyard.

- File location: Game/Scripts/common.lua
- Loaded from: Game/Scripts/main.lua

Globals

Use the following globals to avoid temporary Lua memory allocations:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_SignalData_point</td>
<td>Basic 3D vector value used by g_SignalData.</td>
</tr>
<tr>
<td>g_SignalData_point2</td>
<td>Basic 3D vector value used by g_SignalData.</td>
</tr>
<tr>
<td>g_SignalData</td>
<td>Used to pass signal data in AI behavior scripts. See Signals.</td>
</tr>
<tr>
<td>g_StringTemp1</td>
<td>Commonly used for temporary strings inside Lua functions.</td>
</tr>
<tr>
<td>g_HitTable</td>
<td>Commonly used by the Physics.RaycastWorldIntersection function.</td>
</tr>
</tbody>
</table>

A g_HitTable used with Physics.RaycastWorldIntersection can contain the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>3D vector world coordinates of the ray hit.</td>
</tr>
<tr>
<td>normal</td>
<td>3D normal vector of the ray hit.</td>
</tr>
<tr>
<td>dist</td>
<td>Distance of the ray hit.</td>
</tr>
<tr>
<td>surface</td>
<td>Type of surface hit.</td>
</tr>
<tr>
<td>entity</td>
<td>Script table of entity hit (if one was hit).</td>
</tr>
<tr>
<td>renderNode</td>
<td>Script handle to a foliage or static render node.</td>
</tr>
</tbody>
</table>

A g_SignalData table can contain the following parameter types:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vec3</td>
<td>3D vector.</td>
</tr>
<tr>
<td>ScriptHandle</td>
<td>Normally used to pass along an entity ID.</td>
</tr>
<tr>
<td>Floating Point</td>
<td>Floating point value.</td>
</tr>
<tr>
<td>Integer</td>
<td>Integer or number value.</td>
</tr>
<tr>
<td>String</td>
<td>String value.</td>
</tr>
</tbody>
</table>
**AIReload()**

Reloads the `aiconfig.lua` Lua script (`Game/Scripts/AI/`).

**AIDebugToggle()**

Toggles the `ai_DebugDraw` console variable on and off.

**ShowTime()**

Logs the current system time to the console. Format is Day/Month/Year, Hours:Minutes.

**count()**

Returns the number of key-value pairs in a given table.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_tbl</td>
<td>Table to retrieve the number of key-value pairs from.</td>
</tr>
</tbody>
</table>

**new()**

Creates a new table by copying an specified existing table. This function is commonly used to create a local table based on an entity parameter table.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_obj</td>
<td>Existing table you want to create a new one from.</td>
</tr>
<tr>
<td>norecurse</td>
<td>Flag indicating whether or not to recursively recreate all sub-tables. If set to TRUE, sub-tables will not be recreated.</td>
</tr>
</tbody>
</table>

**merge()**

Merges two tables without merging functions from the source table.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst</td>
<td>Destination table to merge source table information into.</td>
</tr>
<tr>
<td>src</td>
<td>Source table to merge table information from.</td>
</tr>
<tr>
<td>recurse</td>
<td>Flag indicating whether or not to recursively merge all sub-tables.</td>
</tr>
</tbody>
</table>

**mergef()**

Merges two tables including merging functions from the source table.
Vec2Str()

Converts a 3D vector table into a string and returns it in the following format: (x: X.XXX y: Y.YYY z: Z.ZZZ).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst</td>
<td>Destination table to merge source table information into.</td>
</tr>
<tr>
<td>src</td>
<td>Source table to merge table information from.</td>
</tr>
<tr>
<td>recursive</td>
<td>Flag indicating whether or not to recursively merge all sub-tables.</td>
</tr>
</tbody>
</table>

LogError()

Logs an error message to the console and the log file. Message appears in red text in the console.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmt</td>
<td>Formatted message string.</td>
</tr>
<tr>
<td>...</td>
<td>Optional argument list. For example: LogError(&quot;MyError: %f&quot;, math.pi);</td>
</tr>
</tbody>
</table>

LogWarning()

Logs a warning message to the console and the log file. Message appears in yellow text in the console.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmt</td>
<td>Formatted message string.</td>
</tr>
<tr>
<td>...</td>
<td>Optional argument list. For example: LogWarning(&quot;MyError: %f&quot;, math.pi);</td>
</tr>
</tbody>
</table>

Log()

Logs a message to the console and the log file. Commonly used for debugging purposes.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmt</td>
<td>Formatted message string.</td>
</tr>
<tr>
<td>...</td>
<td>Optional argument list. For example: Log(&quot;MyLog: %f&quot;, math.pi);</td>
</tr>
</tbody>
</table>
### dump()

Dumps information from a specified table to the console.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_class</td>
<td>Table to dump to console. For example: g_localActor</td>
</tr>
<tr>
<td>no_func</td>
<td>Flag indicating whether or not to dump the table functions.</td>
</tr>
<tr>
<td>depth</td>
<td>Depth of the tables tree dump information from.</td>
</tr>
</tbody>
</table>

### EmptyString()

Checks whether or not a given string is set and its length is greater than zero. Returns **TRUE** or **FALSE**.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>String to check for.</td>
</tr>
</tbody>
</table>

### NumberToBool()

Checks whether or not a number value is true (non-zero) or false (zero).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Number to check for.</td>
</tr>
</tbody>
</table>

### EntityName()

Retrieves the name of a specified entity ID or entity table. If the entity doesn't exist, this function returns an empty string.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>Entity table or entity ID to return a name for.</td>
</tr>
</tbody>
</table>

### EntityNamed()

Checks whether or not an entity with the specified name exists in the entity system. Returns **TRUE** or **FALSE**. Commonly used for debugging.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of entity to check for.</td>
</tr>
</tbody>
</table>
SafeTableGet()

Checks whether or not a sub-table with a specified name exists in a table. If the sub-table exists, this function returns it; otherwise the function returns `nil`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>Table to check for the existence of a sub-table.</td>
</tr>
<tr>
<td>name</td>
<td>Sub-table name to check for.</td>
</tr>
</tbody>
</table>

EntityUtils Lua Functions

CryLua is deprecated and will be removed in a future version of Lumberyard.

This topic describes the commonly used Lua entity utility functions.

- **File location:** Game/Scripts/Utils/EntityUtils.lua
- **Loaded from:** Game/Scripts/common.lua

DumpEntities()

Dumps to console all entity IDs, names, classes, positions, and angles that are currently used in a map. For example:

```
[userdata: 00000002].name=Grunt1 clsid=Grunt pos=1016.755,1042.764,100.000
  ang=0.000,0.000,1.500
[userdata: 00000003].name=Grunt2 clsid=Grunt pos=1020.755,1072.784,100.000
  ang=0.000,0.000,0.500
...
```

CompareEntitiesByName()

Compares two entities identified by name. This function is commonly used when sorting tables.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ent1</td>
<td>Name of first entity table.</td>
</tr>
<tr>
<td>ent2</td>
<td>Name of second entity table.</td>
</tr>
</tbody>
</table>

Example

```
local entities = System.GetEntitiesByClass("SomeEntityClass")
table.sort(entities, CompareEntitiesByName)
```
CompareEntitiesByDistanceFromPoint()

Compares the distance of two entities from a specified point. If the distance is greater for Entity 1 than for Entity 2 (that is, Entity 1 is further away), this function returns TRUE, otherwise it returns FALSE.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ent1</td>
<td>Entity 1 table</td>
</tr>
<tr>
<td>ent2</td>
<td>Entity 2 table</td>
</tr>
<tr>
<td>point</td>
<td>3D position vector identifying the point to measure distance to.</td>
</tr>
</tbody>
</table>

Example

```lua
local ent1 = System.GetEntityByName("NameEntityOne")
local ent2 = System.GetEntityByName("NameEntityTwo")

if(CompareEntitiesByDistanceFromPoint( ent1, ent2, g_localActor:GetPos()))then
    Log("Entity One is further away from the Player than Entity two...")
end
```

BroadcastEvent()

Processes an entity event broadcast.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sender</td>
<td>Entity that sent the event.</td>
</tr>
<tr>
<td>event</td>
<td>String based entity event to process.</td>
</tr>
</tbody>
</table>

Example

```lua
BroadcastEvent(self, "Used")
```

MakeDerivedEntity()

Creates a new table that is a derived version of a parent entity table. This function is commonly used to simplify the creation of a new entity script based on another entity.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_DerivedClass</td>
<td>Derived class table.</td>
</tr>
<tr>
<td>_Parent</td>
<td>Parent or base class table.</td>
</tr>
</tbody>
</table>
MakeDerivedEntityOverride()

Creates a new table that is a derived class of a parent entity. The derived table's properties will override those from the parent.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_DerivedClass</td>
<td>Derived class table.</td>
</tr>
<tr>
<td>_Parent</td>
<td>Parent or base class table.</td>
</tr>
</tbody>
</table>

MakeUsable()

Adds usable functionality, such as an OnUsed event, to a specified entity.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>Entity table to make usable.</td>
</tr>
</tbody>
</table>

Example

```lua
MyEntity = { ... whatever you usually put here ... }
MakeUsable(MyEntity)

function MyEntity:OnSpawn() ...
function MyEntity:OnReset()
    self:ResetOnUsed()
    ...
end
```

MakePickable()

Adds basic "pickable" functionality to a specified entity. The bPickable property is added to the entity's properties table.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>Entity table to make pickable.</td>
</tr>
</tbody>
</table>

MakeSpawnable()

Adds spawn functionality to a specified entity. Commonly used for AI actors during creation.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>Entity table to make spawnable.</td>
</tr>
</tbody>
</table>
EntityCommon.PhysicalizeRigid()

Physicalizes an entity based on the specified entity slot and its physics properties.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>Entity table to physicalize.</td>
</tr>
<tr>
<td>nSlot</td>
<td>Entity slot to physicalize.</td>
</tr>
<tr>
<td>Properties</td>
<td>Physics properties table</td>
</tr>
<tr>
<td>bActive</td>
<td>Not used.</td>
</tr>
</tbody>
</table>

Lua Vector and Math Functions

CryLua is deprecated and will be removed in a future version of Lumberyard.

This topic describes the commonly used math global vectors, constants, and functions.

- File location: Game/Scripts/Utils/Math.lua
- Loaded from: Game/Scripts/common.lua

Global Vectors

The following globals should be used to avoid temporary Lua memory allocations:

<table>
<thead>
<tr>
<th>Global Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_Vectors.v000</td>
<td>Basic zero vector.</td>
</tr>
<tr>
<td>g_Vectors.v001</td>
<td>Positive z-axis direction vector.</td>
</tr>
<tr>
<td>g_Vectors.v010</td>
<td>Positive y-axis direction vector.</td>
</tr>
<tr>
<td>g_Vectors.v100</td>
<td>Positive x-axis direction vector.</td>
</tr>
<tr>
<td>g_Vectors.v101</td>
<td>The x and z-axis direction vector.</td>
</tr>
<tr>
<td>g_Vectors.v110</td>
<td>The x and y-axis direction vector.</td>
</tr>
<tr>
<td>g_Vectors.v111</td>
<td>The x, y and z-axis vector.</td>
</tr>
<tr>
<td>g_Vectors.up</td>
<td>Positive z-axis direction vector.</td>
</tr>
<tr>
<td>g_Vectors.down</td>
<td>Negative z-axis direction vector.</td>
</tr>
<tr>
<td>g_Vectors.temp</td>
<td>Temporary zero vector.</td>
</tr>
<tr>
<td>g_Vectors.tempColor</td>
<td>Temporary zero vector. Commonly used for passing rgb color values.</td>
</tr>
<tr>
<td>g_Vectors.temp_v1</td>
<td>Temporary zero vector.</td>
</tr>
</tbody>
</table>
### Global Name

<table>
<thead>
<tr>
<th>Global Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_Vectors.temp_v2</td>
<td>Temporary zero vector.</td>
</tr>
<tr>
<td>g_Vectors.temp_v3</td>
<td>Temporary zero vector.</td>
</tr>
<tr>
<td>g_Vectors.temp_v4</td>
<td>Temporary zero vector.</td>
</tr>
<tr>
<td>g_Vectors.vecMathTemp1</td>
<td>Temporary zero vector.</td>
</tr>
<tr>
<td>g_Vectors.vecMathTemp2</td>
<td>Temporary zero vector.</td>
</tr>
</tbody>
</table>

### Constants

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g_Rad2Deg</td>
<td>Basic radian-to-degree conversion value.</td>
</tr>
<tr>
<td>g_Deg2Rad</td>
<td>Basic degree-to-radian conversion value.</td>
</tr>
<tr>
<td>g_Pi</td>
<td>Basic Pi constant based on math.pi.</td>
</tr>
<tr>
<td>g_2Pi</td>
<td>Basic double-Pi constant based on math.pi.</td>
</tr>
<tr>
<td>g_Pi2</td>
<td>Basic half-Pi constant based on math.pi.</td>
</tr>
</tbody>
</table>

### IsNullVector()

Checks whether or not all components of a specified vector are null.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Vector to check.</td>
</tr>
</tbody>
</table>

### IsNotNullVector()

Checks whether or not any components of a specified vector is not null.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Vector to check.</td>
</tr>
</tbody>
</table>

### LengthSqVector()

Retrieves the squared length of a specified vector.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Vector to retrieve length for.</td>
</tr>
</tbody>
</table>
**LengthVector()**

Retrieves the length of a specified vector.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Vector to retrieve length for.</td>
</tr>
</tbody>
</table>

**DistanceSqVectors()**

Retrieves the squared distance between two vectors.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>First vector.</td>
</tr>
<tr>
<td>b</td>
<td>Second vector.</td>
</tr>
</tbody>
</table>

**DistanceSqVectors2d()**

Retrieves the squared distance between two vectors in 2D space (without z-component).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>First vector.</td>
</tr>
<tr>
<td>b</td>
<td>Second vector.</td>
</tr>
</tbody>
</table>

**DistanceVectors()**

Retrieves the distance between two vectors.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>First vector.</td>
</tr>
<tr>
<td>b</td>
<td>Second vector.</td>
</tr>
</tbody>
</table>

**dotproduct3d()**

Retrieves the dot product between two vectors.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>First vector.</td>
</tr>
<tr>
<td>b</td>
<td>Second vector.</td>
</tr>
</tbody>
</table>
dotproduct2d()

Retrieves the dot product between two vectors in 2D space (without z-component).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>First vector.</td>
</tr>
<tr>
<td>b</td>
<td>Second vector.</td>
</tr>
</tbody>
</table>

LogVec()

Logs a specified vector to console.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Descriptive name of the vector.</td>
</tr>
<tr>
<td>v</td>
<td>Vector to log.</td>
</tr>
</tbody>
</table>

Example

LogVec("Local Actor Position", g_localActor:GetPos())

Console output:

<Lua> Local Actor Position = (1104.018066 1983.247925 112.769440)

ZeroVector()

Sets all components of a specified vector to zero.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest</td>
<td>Vector to zero out.</td>
</tr>
</tbody>
</table>

CopyVector()

Copies the components of one vector to another.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest</td>
<td>Destination vector.</td>
</tr>
<tr>
<td>src</td>
<td>Source vector.</td>
</tr>
</tbody>
</table>
SumVectors()

Adds up the components of two vectors.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>First vector.</td>
</tr>
<tr>
<td>b</td>
<td>Second vector.</td>
</tr>
</tbody>
</table>

NegVector()

Negates all components of a specified vector.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Vector to negate.</td>
</tr>
</tbody>
</table>

SubVectors()

Copies the componentwise subtraction of two vectors to a destination vector.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest</td>
<td>Destination vector.</td>
</tr>
<tr>
<td>a</td>
<td>First vector.</td>
</tr>
<tr>
<td>b</td>
<td>Second vector.</td>
</tr>
</tbody>
</table>

FastSumVectors()

Copies the componentwise addition of two vectors to a destination vector.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest</td>
<td>Destination vector.</td>
</tr>
<tr>
<td>a</td>
<td>First vector.</td>
</tr>
<tr>
<td>b</td>
<td>Second vector.</td>
</tr>
</tbody>
</table>

DifferenceVectors()

Retrieves the difference between two vectors.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>First vector.</td>
</tr>
</tbody>
</table>
### FastDifferenceVectors()

Copies the componentwise difference between two vectors to a destination vector.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest</td>
<td>Destination vector.</td>
</tr>
<tr>
<td>a</td>
<td>First vector.</td>
</tr>
<tr>
<td>b</td>
<td>Second vector.</td>
</tr>
</tbody>
</table>

### ProductVectors()

Retrieves the product of two vectors.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>First vector.</td>
</tr>
<tr>
<td>b</td>
<td>Second vector.</td>
</tr>
</tbody>
</table>

### FastProductVectors()

Copies the product of two vectors to a destination vector.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest</td>
<td>Destination vector.</td>
</tr>
<tr>
<td>a</td>
<td>First vector.</td>
</tr>
<tr>
<td>b</td>
<td>Second vector.</td>
</tr>
</tbody>
</table>

### ScaleVector()

Scales a specified vector \( a \) by a factor of \( b \).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Vector.</td>
</tr>
<tr>
<td>b</td>
<td>Scalar.</td>
</tr>
</tbody>
</table>
**ScaleVectorInPlace(a,b)**

Retrieves a new vector based on a copy of vector \( a \) scaled by a factor \( b \).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>First vector.</td>
</tr>
<tr>
<td>b</td>
<td>Scalar.</td>
</tr>
</tbody>
</table>

**ScaleVectorInPlace(dest,a,b)**

Copies vector \( a \) scaled by the factor of \( b \) to a destination vector.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest</td>
<td>Destination vector.</td>
</tr>
<tr>
<td>a</td>
<td>First vector.</td>
</tr>
<tr>
<td>b</td>
<td>Scalar.</td>
</tr>
</tbody>
</table>

**NormalizeVector()**

Normalizes a specified vector.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Vector to normalize.</td>
</tr>
</tbody>
</table>

**VecRotate90_Z()**

Rotates a specified vector by 90 degree around the z-axis.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>Vector to rotate.</td>
</tr>
</tbody>
</table>

**VecRotateMinus90_Z()**

Rotates a specified vector by -90 degree around the z-axis.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>Vector to rotate.</td>
</tr>
</tbody>
</table>
crossproduct3d()
Copies the result of the cross product between two vectors to a destination vector.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest</td>
<td>Destination vector.</td>
</tr>
<tr>
<td>p</td>
<td>First vector.</td>
</tr>
<tr>
<td>q</td>
<td>Second vector.</td>
</tr>
</tbody>
</table>

RotateVectorAroundR()
Copies to a destination vector the result of the vector rotation of vector \( p \) around vector \( r \) by a specified angle.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest</td>
<td>Destination vector.</td>
</tr>
<tr>
<td>p</td>
<td>First vector.</td>
</tr>
<tr>
<td>r</td>
<td>Second vector.</td>
</tr>
<tr>
<td>angle</td>
<td>Rotation angle.</td>
</tr>
</tbody>
</table>

ProjectVector()
Copies to a destination vector the result of the vector projection of vector \( P \) to the surface with a specified normal \( N \).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest</td>
<td>Destination vector.</td>
</tr>
<tr>
<td>P</td>
<td>Vector to project.</td>
</tr>
<tr>
<td>N</td>
<td>Surface normal.</td>
</tr>
</tbody>
</table>

DistanceLineAndPoint()
Retrieves the distance between point \( a \) and the line between \( p \) and \( q \).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Point to measure from.</td>
</tr>
<tr>
<td>p</td>
<td>Vector ( p ).</td>
</tr>
<tr>
<td>q</td>
<td>Vector ( q ).</td>
</tr>
</tbody>
</table>
LerpColors()

Performs linear interpolation between two color/vectors with a factor of \( k \).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Color/vector a.</td>
</tr>
<tr>
<td>b</td>
<td>Color/vector b.</td>
</tr>
<tr>
<td>k</td>
<td>Factor k.</td>
</tr>
</tbody>
</table>

Lerp()

Performs linear interpolation between two scalars with a factor of \( k \).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Scalar a.</td>
</tr>
<tr>
<td>b</td>
<td>Scalar b.</td>
</tr>
<tr>
<td>k</td>
<td>Factor k.</td>
</tr>
</tbody>
</table>

__max()  

Retrieves the maximum of two scalars.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Scalar a.</td>
</tr>
<tr>
<td>b</td>
<td>Scalar b.</td>
</tr>
</tbody>
</table>

__min()  

Retrieves the minimum of two scalars.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Scalar a.</td>
</tr>
<tr>
<td>b</td>
<td>Scalar b.</td>
</tr>
</tbody>
</table>

clamp()

Clamps a specified number between minimum and maximum.
### Interpolate()

Interpolates a number to a specified goal by a specified speed.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>actual</code></td>
<td>Number to interpolate.</td>
</tr>
<tr>
<td><code>goal</code></td>
<td>Goal.</td>
</tr>
<tr>
<td><code>speed</code></td>
<td>Interpolation speed.</td>
</tr>
</tbody>
</table>

### sgn()

Retrieves the sign of a specified number (0 returns 0).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a</code></td>
<td>Number to get sign for.</td>
</tr>
</tbody>
</table>

### sgnnz()

Retrieves the sign of a specified number (0 returns 1).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a</code></td>
<td>Number to get sign for.</td>
</tr>
</tbody>
</table>

### sqr()

Retrieves the square of a specified number.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a</code></td>
<td>Number to square.</td>
</tr>
</tbody>
</table>

### randomF()

Retrieves a random float value between two specified numbers.
### iff()

Checks the condition of a test value and returns one of two other values depending on whether the test value is `nil` or not.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Test value.</td>
</tr>
<tr>
<td>a</td>
<td>Return value if test value is not nil.</td>
</tr>
<tr>
<td>b</td>
<td>Return value if test value is nil.</td>
</tr>
</tbody>
</table>

### Physics Lua Functions

CryLua is deprecated and will be removed in a future version of Lumberyard.

These functions are commonly used to register new explosion and crack shapes in the physics engine.

File location: Game/Scripts/physics.lua

- Loaded from: Game/Scripts/main.lua

#### Physics.RegisterExplosionShape()

Registers a boolean carving shape for breakable objects in the physics engine.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sGeometryFile</td>
<td>Name of a boolean shape cgf file.</td>
</tr>
<tr>
<td>fSize</td>
<td>Shape's characteristic size.</td>
</tr>
<tr>
<td>BreakId</td>
<td>Breakability index (0-based) used to identify the breakable material.</td>
</tr>
<tr>
<td>fProbability</td>
<td>Shape's relative probability; when several shapes with the same size appear as candidates for carving, these relative probabilities are used to select one.</td>
</tr>
<tr>
<td>sSplintersfile</td>
<td>Name of a splinters cgf file, used for trees to add splinters at the breakage location.</td>
</tr>
<tr>
<td>fSplintersOffset</td>
<td>Size offset for the splinters.</td>
</tr>
<tr>
<td>sSplintersCloudEffect</td>
<td>Name of splinters particle fx; this effect is played when a splinters-based constraint breaks and splinters disappear.</td>
</tr>
</tbody>
</table>
Physics.RegisterExplosionCrack()

Registers a new explosion crack for breakable objects in the physics engine.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sGeometryFile</td>
<td>Name of a crack shape cgf file. This type of file must have three helpers to mark the corners, named &quot;1&quot;, &quot;2&quot;, and &quot;3&quot;.</td>
</tr>
<tr>
<td>BreakId</td>
<td>Breakability index (0-based) used to identify the breakable material.</td>
</tr>
</tbody>
</table>

Lua ScriptBind Reference

CryLua is deprecated and will be removed in a future version of Lumberyard.

You can use ScriptBind functions in Lua scripts to call legacy code written in C++.

Topics
- ScriptBind Engine Functions (p. 757)
- ScriptBind Action Functions (p. 914)
- ScriptBind_Boids (p. 960)

ScriptBind Engine Functions

CryLua is deprecated and will be removed in a future version of Lumberyard.

Lists C++ engine functions that you can call from Lua script.

Topics
- ScriptBind_AI (p. 757)
- ScriptBind_Entity (p. 825)
- ScriptBind_Movie (p. 877)
- ScriptBind_Particle (p. 878)
- ScriptBind_Physics (p. 881)
- ScriptBind_Script (p. 884)
- ScriptBind_Sound (p. 886)
- ScriptBind_System (p. 888)

ScriptBind_AI

CryLua is deprecated and will be removed in a future version of Lumberyard.

Lists C++ AI functions that can be called from Lua scripts.
**AbortAction**

Aborts execution of a specified action.

**Syntax**

```c
AI.AbortAction(userId [, actionId ])
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>actionId</td>
<td>Unique ID of the action to be aborted. If 0 (or nil), all actions on the</td>
</tr>
<tr>
<td>(optional)</td>
<td>specified entity are aborted.</td>
</tr>
</tbody>
</table>

**AddAggressiveTarget**

Adds a target to a specified entity's list as an aggressive potential target.

**Syntax**

```c
AI.AddAggressiveTarget(entityId, targetId)
```

Returns True if successfully added.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>targetId</td>
<td>Target's entity ID to add.</td>
</tr>
</tbody>
</table>

**AddCombatClass**

Creates new combat class.

**Syntax**

```c
AI.AddCombatClass(int combatClass, SmartScriptTable pTable, const char* szCustomSignal)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>combatClass</td>
<td>Combat class to add.</td>
</tr>
<tr>
<td>pTable</td>
<td>Parameters table.</td>
</tr>
<tr>
<td>szCustomSignal</td>
<td>Specifies optional custom OnSeen signal.</td>
</tr>
</tbody>
</table>

**AddFormationPoint**

Adds a follow-type node to a formation descriptor.

**Syntax**
AI.AddFormationPoint(name, sightangle, distance, offset, [unit_class [,distanceAlt, offsetAlt]])

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the formation descriptor.</td>
</tr>
<tr>
<td>sightangle</td>
<td>Angle of sight of the node (-180,180; 0 = the entity looks forward).</td>
</tr>
<tr>
<td>distance</td>
<td>Distance from the formation's owner.</td>
</tr>
<tr>
<td>offset</td>
<td>Offset along the following line (negative = left, positive = right).</td>
</tr>
<tr>
<td>unit_class</td>
<td>Class of soldier (see eSoldierClass definition in IAgent.h).</td>
</tr>
<tr>
<td></td>
<td>DistanceAlt (optional): alternative distance from the formation owner</td>
</tr>
<tr>
<td></td>
<td>offsetAlt (optional): alternative offset.</td>
</tr>
</tbody>
</table>

**AddFormationPointFixed**

Adds a node with a fixed offset to a formation descriptor.

**Syntax**

AI.AddFormationPointFixed(name, sightangle, x, y, z [,unit_class])

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the formation descriptor.</td>
</tr>
<tr>
<td>sightangle</td>
<td>Angle of sight of the node (-180,180; 0 = the entity looks forward).</td>
</tr>
<tr>
<td>x, y, z</td>
<td>Offset from formation owner.</td>
</tr>
<tr>
<td>unit_class</td>
<td>Class of soldier (see eSoldierClass definition in IAgent.h).</td>
</tr>
</tbody>
</table>

**AddPatternBranch**

Creates a branch pattern at the specified node. When the entity has approached the specified node (nodeName) and it is time to choose a new point, the rules defined by this function are used to select the new point. This function can associate multiple target points and an evaluation rule.

**Syntax**

AI.AddPatternBranch(nodeName, method, branchNode1, branchNode2, ..., branchNodeN)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeName</td>
<td>Name of the node to add branches to.</td>
</tr>
<tr>
<td>method</td>
<td>Method used to choose the next node. Valid values include:</td>
</tr>
<tr>
<td></td>
<td>• AITRACKPAT_CHOOSE_ALWAYS – Choose the next point from the list in linear sequence.</td>
</tr>
</tbody>
</table>
AddPatternNode

Adds a point to the track pattern.

When validating the points, the test is performed from the start position to the end position. Start position is either the pattern origin or, if the parent is provided, the parent position. The end position is either the relative offset from the start position or from the pattern origin; this position is chosen based on the node flag. The offset is clamped to the physical world based on the test method. The points will be evaluated in the same order they are added to the descriptor, and the system does not try to correct the evaluation order. If hierarchies are used (parent name is defined), it is up to the pattern creator to make sure the nodes are created in such an order that the parent is added before it is referenced.

Syntax

AI.AddPatternNode(nodeName, offsetx, offsety, offsetz, flags, [parent], [signalValue])

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeName</td>
<td>Name of the new point. , Point names are local to the current pattern.</td>
</tr>
<tr>
<td>offsetx, offsety, offsetz</td>
<td>Offset from the start position or from the pattern center. See AITRACKPAT_NODE_ABSOLUTE.</td>
</tr>
<tr>
<td>flags</td>
<td>Track pattern functionality flags.</td>
</tr>
<tr>
<td></td>
<td>Node evaluation flags:</td>
</tr>
<tr>
<td></td>
<td>• AITRACKPAT_NODE_START – Node can be used as the first node in the pattern. There can be multiple start nodes. In that case the closest one is chosen.</td>
</tr>
<tr>
<td></td>
<td>• AITRACKPAT_NODE_ABSOLUTE – Interpret offset as an offset from the pattern center (otherwise the offset is from the start position).</td>
</tr>
<tr>
<td></td>
<td>• AITRACKPAT_NODE_SIGNAL – A signal &quot;OnReachedTrackPatternNode&quot; will be sent when the node is reached.</td>
</tr>
<tr>
<td></td>
<td>• AITRACKPAT_NODE_STOP – Advancing will be stopped. It can be continued by calling entity:ChangeAIParameter(AIPARAM_TRACKPATTERN_ADVANCE, 1).</td>
</tr>
<tr>
<td></td>
<td>• AITRACKPAT_NODE_DIRBRANCH – For the direction at each pattern node, use the average direction to the branch nodes (otherwise use the direction from the node position to the center of the pattern).</td>
</tr>
<tr>
<td>parent (optional)</td>
<td>Parent node position, which will be used as the start position instead of the pattern center.</td>
</tr>
</tbody>
</table>
**AddPersonallyHostile**

**Syntax**

AI.AddPersonallyHostile(ScriptHandle entityID, ScriptHandle hostileID)

**AgentLookAtPos**

Causes the specified entity to look at a certain position.

**Syntax**

AI.AgentLookAtPos(entityId, Vec3 pos)

**AllowLowerBodyToTurn**

**Syntax**

AI.AllowLowerBodyToTurn(entityID, bAllowLowerBodyToTurn)

**BeginTrackPattern**

 Begins the definition of a new track pattern descriptor. The pattern is created by calling AI.AddPatternPoint() and AI.AddPatternBranch(), and finalized by calling AI.EndTrackPattern().

**Syntax**

AI.BeginTrackPattern(patternName, flags, validationRadius, [stateTresholdMin],

**BeginTrackPattern**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>patternName</td>
<td>Name of the new track pattern descriptor.</td>
</tr>
</tbody>
</table>
### Parameter Description

**flags**

Track pattern functionality flags.

Validation flags describe how the pattern is validated to fit the physical world:

- **AITRACKPAT_VALIDATE_NONE** – No validation.
- **AITRACKPAT_VALIDATE_SWEPTSPHERE** – Validate using swept sphere tests, where the sphere radius equals the validation radius plus the entity pass radius.
- **AITRACKPAT_VALIDATE_RAYCAST** – Validate using raycasting, where the hit position is pulled back by the amount of validation radius plus the entity pass radius.

Alignment flags describe how, when the pattern is selected to be used, the alignment of the pattern can be changed. Flags are evaluated in the following order:

- **AITRACKPAT_ALIGN_TO_TARGET** – Align the pattern so that the y-axis points towards the target each time it is set. If the agent does not have a valid attention target at the time, the pattern is aligned to the world.
- **AITRACKPAT_ALIGN_RANDOM** – Align the pattern randomly each time it is set. The rotation ranges are set using `SetRandomRotation()`.

**validationRadius**

Validation radius is added to the entity pass radius when validating the pattern along the offsets.

**stateTresholdMin** (optional)

If the state of the pattern is 'enclosed' (high deformation) and the global deformation < stateTresholdMin, the state becomes exposed. Default 0.35.

**stateTresholdMax** (optional)

If the state of the pattern is 'exposed' (low deformation) and the global deformation > stateTresholdMax, the state becomes enclosed. Default 0.4.

**globalDeformTreshold** (optional)

Deformation of the whole pattern is tracked in range [0..1]. This threshold value can be used to clamp the bottom range, so that values in range [trhd..1] becomes [0..1], default 0.0.

**localDeformTreshold** (optional)

Deformation of the each node is tracked in range [0..1]. This threshold value can be used to clamp the bottom range, so that values in range [trhd..1] becomes [0..1], default 0.0.

**exposureMod** (optional)

Importance of the node exposure (how much it is seen by the tracked target) to consider when branching. Valid range is [-1..1], where -1 means to favor unseen nodes and 1 means to favor seen, exposed nodes. Default is 0 (no effect).

**randomRotAng** (optional)

Flag indicating whether or not to randomly rotate the pattern each time it is set. Rotation is performed in XYZ order. This parameter defines angles (in degrees) around each axis.

### CanFireInStance

**Syntax**

```
AI.CanFireInStance(entityId, stance)
```
Returns true if AI can fire at his target in the specified stance at his current position

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>stance.</td>
<td>Stance Id (STANCE_*)</td>
</tr>
</tbody>
</table>

CanMelee

Determines whether or not the AI is able to do melee attack.

Syntax

```plaintext
AI.CanMelee(entityId)
```

Returns True or false (1 or 0).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

CanMoveStraightToPoint

Determines whether or not a specified entity can move in a straight line from its current position to a specified point.

Syntax

```plaintext
AI.CanMove StraightToPoint(entityId, position)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>position</td>
<td>Position to check path to.</td>
</tr>
</tbody>
</table>

ChangeFormation

Changes the formation descriptor for the current formation of a specified entity's group (if one exists).

Syntax

```plaintext
AI.ChangeFormation(entityId, name [,scale])
```

Returns True if the formation change was successful.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>Unique entity ID used to identify the group.</td>
</tr>
</tbody>
</table>
### ChangeMovementAbility

Changes the value of an AI movement ability parameter for the entity specified.

**Syntax**

```c
AI.ChangeMovementAbility(entityId, paramEnum, paramValue)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>paramEnum</td>
<td>Index of the parameter to change. Valid values include:</td>
</tr>
<tr>
<td></td>
<td>• <code>AIMOVEABILITY_OPTIMALFLIGHTHEIGHT</code> – Optimal flight height in meters while finding path.</td>
</tr>
<tr>
<td></td>
<td>• <code>AIMOVEABILITY_MINFLIGHTHEIGHT</code> – Minimum flight height in meters while finding path.</td>
</tr>
<tr>
<td></td>
<td>• <code>AIMOVEABILITY_MAXFLIGHTHEIGHT</code> – Maximum flight height in meters while finding path.</td>
</tr>
<tr>
<td>paramValue</td>
<td>New value for the specified parameter.</td>
</tr>
</tbody>
</table>

### ChangeParameter

Updates a parameter value for a specified entity.

**Syntax**

```c
AI.ChangeParameter(entityId, paramEnum, paramValue)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>paramEnum</td>
<td>The enum of the parameter.</td>
</tr>
<tr>
<td>paramValue</td>
<td>The new value for the specified parameter.</td>
</tr>
</tbody>
</table>

### CheckForFriendlyAgentsAroundPoint

**Syntax**

```c
AI.CheckForFriendlyAgentsAroundPoint(ScriptHandle entityId, Vec3 point, float radius)
```
CheckMeleeDamage
Determines whether or not the AI performing melee is actually hitting target.

Syntax

```
AI.CheckMeleeDamage(entityId, targetId, radius, minheight, maxheight, angle)
```

Returns (distance, angle) pair between entity and target (degrees) if melee is possible, nil otherwise

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>targetId</td>
<td>Target's entity ID.</td>
</tr>
<tr>
<td>radius</td>
<td>max distance in 2d to target.</td>
</tr>
<tr>
<td>minheight</td>
<td>min distance in height.</td>
</tr>
<tr>
<td>maxheight</td>
<td>max distance in height.</td>
</tr>
<tr>
<td>angle</td>
<td>FOV to include target.</td>
</tr>
</tbody>
</table>

ClearAnimationTag

Syntax

```
AI.ClearAnimationTag(ScriptHandle entityId, const char* tagName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>AI's entity.</td>
</tr>
<tr>
<td>tagName</td>
<td></td>
</tr>
</tbody>
</table>

ClearMovementContext

Resets the specified entity's movement context.

Syntax

```
AI.ClearMovementContext(entityId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>context</td>
<td>context value.</td>
</tr>
</tbody>
</table>

ClearPotentialTargets

Clears all the potential targets from a specified entity's perception handler.
**Syntax**

```cpp
AI.ClearPotentialTargets(entityId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

**ClearTempTarget**

Removes the specified entity's temporary potential target so that it is no longer considered for target selection.

**Syntax**

```cpp
AI.ClearTempTarget(entityId)
```

Returns True if successfully updated.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

**ConstrainPointInsideGenericShape**

**Syntax**

```cpp
AI.ConstrainPointInsideGenericShape(position, shapeName[, checkHeight])
```

Returns Nearest point inside the specified shape.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>Position to check.</td>
</tr>
<tr>
<td>shapeName</td>
<td>Name of the shape to test (returned by AI.GetEnclosingGenericShapeOfType).</td>
</tr>
<tr>
<td>checkHeight</td>
<td>Flag indicating whether or not to test for shape height. (default=false). If set to true, the test will check the space between shape.aabb.min.z and shape.aabb.min.z+shape.height.</td>
</tr>
</tbody>
</table>

**CreateFormation**

Creates a formation descriptor and adds a fixed node at 0,0,0 (owner's node).

**Syntax**

```cpp
AI.CreateFormation(name)
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the new formation descriptor.</td>
</tr>
</tbody>
</table>

**CreateGroupFormation**

Creates a group formation with leader (or updates leader).

**Syntax**

```c
AI.CreateGroupFormation(entityId, leaderId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>AI's entity.</td>
</tr>
<tr>
<td>leaderId</td>
<td>New leader.</td>
</tr>
</tbody>
</table>

**CreateStimulusEvent**

Creates a target track stimulus event for the specified entity.

**Syntax**

```c
AI.CreateStimulusEvent(ScriptHandle ownerId, ScriptHandle targetId, const char* stimulusName, SmartScriptTable pData)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownerId</td>
<td>Unique ID of the entity that owns and receives the event.</td>
</tr>
<tr>
<td>targetId</td>
<td>Unique ID of the entity that sends the event and becomes the target.</td>
</tr>
<tr>
<td>stimulusName</td>
<td>Name of the stimulus event.</td>
</tr>
<tr>
<td>pData</td>
<td>Event data (see TargetTrackHelpers::SStimulusEvent).</td>
</tr>
</tbody>
</table>

**CreateTempGenericShapeBox**

Creates a temporary box-shaped generic shape. This temporary shape will be destroyed upon an AI system reset.

**Syntax**

```c
AI.CreateTempGenericShapeBox(Vec3 center, float radius, float height, int type)
```

Returns Shape name.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>center</td>
<td>Center point of the box.</td>
</tr>
</tbody>
</table>

767
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>radius</td>
<td>Size of the box in x and y directions.</td>
</tr>
<tr>
<td>height</td>
<td>Height of the box.</td>
</tr>
<tr>
<td>type</td>
<td>Box shape type (AIAnchor).</td>
</tr>
</tbody>
</table>

**DebugReportHitDamage**

Creates a debug report for the hit damage.

**Syntax**

```c
AI.DebugReportHitDamage(pVictimEntity, pShooterEntity)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pVictimEntity</td>
<td>Victim ID.</td>
</tr>
<tr>
<td>pShooterEntity</td>
<td>Shooter ID.</td>
</tr>
</tbody>
</table>

**DestroyAllTPSQueries**

Destroys all the tactical point system queries.

**Syntax**

```c
AI2.DestroyAllTPSQueries()
```

**DistanceToGenericShape**

**Syntax**

```c
AI.DistanceToGenericShape(Vec3 position, const char* shapeName[, int checkHeight])
```

Returns True if the point is inside the specified shape.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>Position to check.</td>
</tr>
<tr>
<td>shapeName</td>
<td>Name of the shape to test (returned by AI.GetEnclosingGenericShapeOfType).</td>
</tr>
<tr>
<td>checkHeight</td>
<td>Flag indicating whether or not to test for shape height. (default=false). If set to true, the test will check the space between shape.aabb.min.z and shape.aabb.min.z+shape.height.</td>
</tr>
</tbody>
</table>

**DropTarget**

Clears the target from a specified entity's perception handler.
### Syntax

```c
AI.DropTarget(entityId, targetId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>targetId</td>
<td>Target's entity ID.</td>
</tr>
</tbody>
</table>

### EnableCoverFire

Enables or disables fire when the `FIREMODE_COVER` is selected.

**Syntax**

```c
AI.EnableCoverFire(entityId, enable)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>enable</td>
<td>Boolean.</td>
</tr>
</tbody>
</table>

### EnableFire

Enables or disables fire.

**Syntax**

```c
AI.EnableFire(entityId, enable)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>enable</td>
<td>Boolean.</td>
</tr>
</tbody>
</table>

### EnableUpdateLookTarget

**Syntax**

```c
AI.EnableUpdateLookTarget(ScriptHandle entityId, bool bEnable)
```

### EnableWeaponAccessory

Enables or disables certain weapon accessory usage.

**Syntax**
AI.EnableWeaponAccessory(entityId, int accessory, bool state)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>accessory</td>
<td>Enum of the accessory to enable. Possible values (see enum EAIWeaponAccessories in the IAgent.h file):</td>
</tr>
<tr>
<td>state</td>
<td>Set to true or false to enable or disable.</td>
</tr>
</tbody>
</table>

AIWEPA_NONE = 0,
AIWEPA_LASER = 0x0001
AIWEPA_COMBAT_LIGHT = 0x0002
AIWEPA_PATROL_LIGHT = 0x0004

EndTrackPattern

Finalizes the track pattern definition. This function should always called to finalize the pattern. Failing to do so will cause erratic behavior.

Syntax

AI.EndTrackPattern()

Error

The fallback error message used when the system experiences an unhandled exception. The code following should continue if it is running in the editor so that the original cause of the problem can be fixed, but halt execution when it is running in the game.

Syntax

AI.Error(szMessage)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>szMessage</td>
<td>The message to write to the log.</td>
</tr>
</tbody>
</table>

EvalPeek

Determines whether or not specified entity can peek from its current position.

Syntax

AI.EvalPeek(entityId [, bGetOptimalSide])

Returns One of the following values:

- -1 – don't need to peek
- 0 – cannot peek
- 1 – can peek from left
- 2 – can peek from right
- 3 – can peek from left & right

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>bGetOptimalSide</td>
<td>Flag indicating whether or not to return the side that best fits the attention target's current location, if the AI object can peek from both sides. The default is false.</td>
</tr>
</tbody>
</table>

**ExecuteAction**

Executes an action on a set of participants.

**Syntax**

```
AI.ExecuteAction(action, participant1 [, participant2 [, ... , participantN ]])
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>The smart object action name or ID.</td>
</tr>
<tr>
<td>participant1</td>
<td>The entity ID of the first participant in the action.</td>
</tr>
<tr>
<td>participant2..N (optional)</td>
<td>The entity ID of additional participants.</td>
</tr>
</tbody>
</table>

**FindObjectOfType**

Searches for the closest AIObject of a specified type in an area around a specified entity or position. Once an AIObject is found, it is devalued and can't be found again for a certain number of seconds (unless turned off in flags).

**Syntax**

```
AI.FindObjectOfType(entityId, radius, AIObjectType, flags [,returnPosition [,returnDirection]])
AI.FindObjectOfType(position, radius, AIObjectType, [,returnPosition [,returnDirection]])
```

Returns The found AIObject's name.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>Unique entity ID used to determine the center position of the search.</td>
</tr>
<tr>
<td>position</td>
<td>Vector specifying the center position of the search.</td>
</tr>
<tr>
<td>radius</td>
<td>Radius of the search area.</td>
</tr>
</tbody>
</table>
### FindStandbySpotInShape

**Syntax**

```c
AI.FindStandbySpotInShape(centerPos, targetPos, anchorType)
```

### FindStandbySpotInSphere

**Syntax**

```c
AI.FindStandbySpotInSphere(centerPos, targetPos, anchorType)
```

### FreeSignal

Sends a signal to anyone in a specified radius around a position.

**Syntax**

```c
AI.FreeSignal(signalType, signalText, position, radius [, entityID [,signalExtraData]])
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>signalType</td>
<td>See AI.Signal.</td>
</tr>
<tr>
<td>signalText</td>
<td>See AI.Signal.</td>
</tr>
<tr>
<td>position</td>
<td>The center point {(x,y,z) coordinates} from which the signal is sent.</td>
</tr>
<tr>
<td>radius</td>
<td>The inside radius of the area to which the signal is sent.</td>
</tr>
<tr>
<td>entityID</td>
<td>Optional. The ID of an entity that is a member of a group that should not receive the signal. Entities whose group ID is the value specified will not be sent the signal.</td>
</tr>
</tbody>
</table>
### SignalExtraData

Optional. See `AI.Signal`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>signalExtraData</code></td>
<td>Optional. See <code>AI.Signal</code>.</td>
</tr>
</tbody>
</table>

## GetAIObjectPosition

Retrieves a specified AIObject's position.

### Syntax

```
AI.GetAIObjectPosition(entityId | AIObjectName)
```

Returns AI Object position vector \((x,y,z)\).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>entityId</code></td>
<td>Unique entity ID or AIObject name.</td>
</tr>
</tbody>
</table>

## GetAnchor

Searches for the closest anchor of a specified type in an area around a specified entity. Once an anchor is found, it is devalued and can't be found again for a certain number of seconds (unless turned off in flags).

### Syntax

```
AI.GetAnchor(entityId, radius, AIAnchorType, searchType [,returnPosition [,returnDirection]])
```

Returns The found anchor's name.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>entityId</code></td>
<td>Unique entity ID used to determine the center position of the search.</td>
</tr>
<tr>
<td><code>radius</code></td>
<td>Radius of the search area. Alternatively a search range can be specified (min=minRad,max=maxRad).</td>
</tr>
<tr>
<td><code>AIAnchorType</code></td>
<td>Anchor type to search for. See <code>Scripts/AIAnchor.lua</code> for a complete list of anchor types available.</td>
</tr>
<tr>
<td><code>searchType</code></td>
<td>A combination of one or more of the following search filter flags:</td>
</tr>
<tr>
<td></td>
<td>• <code>AIANCHOR_NEAREST</code> – Nearest anchor of the specified type (default).</td>
</tr>
<tr>
<td></td>
<td>• <code>AIANCHOR_NEAREST_IN_FRONT</code> – Nearest anchor of the specified type inside the front cone of the entity.</td>
</tr>
<tr>
<td></td>
<td>• <code>AIANCHOR_NEAREST_FACING_AT</code> – Nearest anchor of the specified type that is oriented towards entity's attention target</td>
</tr>
<tr>
<td></td>
<td>• <code>AIANCHOR_RANDOM_IN_RANGE</code> – Random anchor of the specified type.</td>
</tr>
<tr>
<td></td>
<td>• <code>AIANCHOR_NEAREST_TO_REFPOINT</code> – Anchor of the specified type that is nearest to the entity's reference point.</td>
</tr>
</tbody>
</table>
### GetAttentionTargetAIType

Retrieves the AI type (AIOBJECT_*) of a specified entity's attention target.

**Syntax**

```c
AI.GetAttentionTargetAIType(entityId)
```

Returns Attention target's AI type, or AIOBJECT_NONE if no target.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

### GetAttentionTargetDirection

Retrieves the direction of a specified entity's attention target.

**Syntax**

```c
AI.GetAttentionTargetDirection(entityId, returnDir)
```

Returns Attention target's direction vector \( \{x,y,z\} \), passed as a return value.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

### GetAttentionTargetDistance

Retrieves the distance from a specified entity to its attention target.

**Syntax**

```c
AI.GetAttentionTargetDistance(entityId)
```

Returns distance to the attention target.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>
GetAttentionTargetEntity

Retrieves a specified entity's attention target entity (if it is a specified entity), or the owner entity of a dummy object's attention target (if there is an owner entity).

**Syntax**

```plaintext
AI.GetAttentionTargetEntity(ScriptHandle entityId)
```

Returns Attention target's entity.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

GetAttentionTargetOf

Retrieves a specified entity's attention target.

**Syntax**

```plaintext
AI.GetAttentionTargetOf(entityId)
```

Returns Name of attention target. Null if there is no target.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

GetAttentionTargetPosition

Retrieves the position of a specified entity's attention target.

**Syntax**

```plaintext
AI.GetAttentionTargetPosition(entityId, returnPos)
```

Returns Attention target 's position vector \(x,y,z\), passed as a return value ().

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

GetAttentionTargetThreat

**Syntax**

```plaintext
AI.GetAttentionTargetThreat(ScriptHandle entityId)
```
GetAttentionTargetType

Retrieves the type (AITARGET_* ) of a specified entity's attention target.

Syntax

\[
\text{AI.GetAttentionTargetType}(\text{entityId})
\]

Returns Attention target's type, or AITARGET_NONE if no target.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

GetAttentionTargetViewDirection

Retrieves the view direction of a specified entity's attention target.

Syntax

\[
\text{AI.GetAttentionTargetViewDirection}(\text{entityId}, \text{returnDir})
\]

Returns Attention target's view direction vector \( \{x,y,z\} \), passed as a return value.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

GetBeaconPosition

Gets the beacon position for a specified entity/object's group.

Syntax

\[
\text{AI.GetBeaconPosition}(\text{entityId | AIObjectName}, \text{returnPos})
\]

Returns True if the beacon is found and the position set.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>AIObjectName</td>
</tr>
<tr>
<td>returnPos</td>
<td>Beacon position vector ( {x,y,z} ).</td>
</tr>
</tbody>
</table>

GetBehaviorBlackBoard

Retrieves a specified AIActor current behaviour's black board (a Lua table).

Syntax
AI.GetBehaviorBlackBoard(entity)

Returns black board – if there was one nil – Otherwise

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId or entityName.</td>
<td>An AIActor identifier.</td>
</tr>
</tbody>
</table>

GetBehaviorVariable

Returns a behavior variable for the specified actor.

Syntax

AI.GetBehaviorVariable(ScriptHandle entityId, const char* variableName)

GetBiasedDirection

Retrieves biased direction of certain point.

Syntax

AI.GetBiasedDirection(entityId)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

GetCurrentHideAnchor

Retrieves the name of the anchor that the entity currently is using for cover.

Syntax

AI.GetCurrentHideAnchor(entityId)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

GetDirectAnchorPos

Retrieves the position of a cover point that a specified entity can use to directly attack its attention target.

Syntax

AI.GetDirectAttackPos(entityId, searchRange, minAttackRange)
Returns Point value, or none if no attack point is available.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>AIAnchorType</td>
<td>Anchor type (see Scripts/AIAnchor.lua for a complete list of anchor types).</td>
</tr>
<tr>
<td>maxDist</td>
<td>Maximum size of search range.</td>
</tr>
</tbody>
</table>

**GetDirLabelToPoint**

Retrieves a direction label (front=0, back=1, left=2, right=3, above=4, -1=invalid) to the specified point.

**Syntax**

`AI.GetDirLabelToPoint(entityId, point)`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>point</td>
<td>The point to evaluate.</td>
</tr>
</tbody>
</table>

**GetEnclosingSpace**

Returns the estimated surrounding navigable space in meters.

**Syntax**

`AI.GetEnclosingSpace(entityId, Vec3 pos, float rad)`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>pos</td>
<td>Check position.</td>
</tr>
<tr>
<td>rad</td>
<td>Check radius.</td>
</tr>
</tbody>
</table>

**GetDistanceAlongPath**

Retrieves the distance between a first and second entity, measured along the first entity's path.

**Syntax**

`AI.GetDistanceAlongPath(entityId1, entityId2)`

Returns Distance along a path. Value can be negative if the second entity is ahead along the path.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId1</td>
<td>ID for the first entity.</td>
</tr>
<tr>
<td>entityId2</td>
<td>ID for the second entity.</td>
</tr>
</tbody>
</table>

**GetDistanceToClosestGroupMember**

**Syntax**

```plaintext
AI.GetDistanceToClosestGroupMember(ScriptHandle entityId)
```

**GetEnclosingGenericShapeOfType**

Retrieves the first shape of a certain type that encloses a specified point.

**Syntax**

```plaintext
AI.GetEnclosingGenericShapeOfType(position, type[, checkHeight])
```

Returns Shape name.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>Point to search for an enclosing shape.</td>
</tr>
<tr>
<td>type</td>
<td>Shape type to search for (uses anchor types).</td>
</tr>
<tr>
<td>checkHeight</td>
<td>(optional) Flag indicating whether or not to test for shape height. (default=false). If set to true, the test checks the space between shape.aabb.min.z and shape.aabb.min.z+shape.height.</td>
</tr>
</tbody>
</table>

**GetExtraPriority**

Retrieves the extra priority value for a specified enemy entity.

**Syntax**

```plaintext
AI.GetExtraPriority(enemyEntityId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enemyEntityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

**GetFactionOf**

Retrieves the faction of the specified entity.

**Syntax**

```plaintext
AI.GetFactionOf(ScriptHandle entityID)
```
Returns the faction of the specified entity.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity whose faction to return.</td>
</tr>
</tbody>
</table>

**GetFormationLookingPoint**

Retrieves the looking point position inside the formation.

**Syntax**

```plaintext
AI.GetFormationLookingPoint(entityId)
```

Returns v3 – table with format {x,y,z} storing the looking point position

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>AI's entity.</td>
</tr>
</tbody>
</table>

**GetFormationPointClass**

Adds a follow-type node to a formation descriptor.

**Syntax**

```plaintext
AI.GetFormationPointClass(name, position)
```

Returns class of formation point (-1 if none found).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the formation descriptor.</td>
</tr>
<tr>
<td>position</td>
<td>Point index in the formation (1..N).</td>
</tr>
</tbody>
</table>

**GetFormationPointPosition**

Retrieves an entity's formation point position.

**Syntax**

```plaintext
AI.GetFormationPointPosition(entityId, pos)
```

Returns true if the formation point has been found.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>
### Parameter Description

| pos | Return value for the position of the entity's formation point. |

#### GetFormationPosition

Retrieves the relative position inside the formation.

**Syntax**

```
AI.GetFormationPosition(entityId)
```

Returns v3 – table with format {x,y,z} storing the relative position

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>AI's entity.</td>
</tr>
</tbody>
</table>

#### GetGroupAveragePosition

Retrieves the average position of a group's members.

**Syntax**

```
AI.GetGroupAveragePosition(entityId, properties, returnPos)
```

Returns the average position.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>Unique entity ID used to determine the group.</td>
</tr>
<tr>
<td>unitProperties</td>
<td>Binary mask of unit properties type for which the attack is requested, in the following form:</td>
</tr>
<tr>
<td></td>
<td>UPR_* + UPR* (UPR_COMBAT_GROUND + UPR_COMBAT_FLIGHT)</td>
</tr>
<tr>
<td></td>
<td>See IAgent.h for a definition of unit properties UPR_.</td>
</tr>
</tbody>
</table>

#### GetGroupCount

Retrieves the member count of a specified entity's group.

**Syntax**

```
AI.GetGroupCount(entityId, flags, type)
```

Returns the count of members for the specified group.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The entity or group ID.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| flags     | A combination of one or more of the following flags:  
  - GROUP_ALL – Counts all agents in the group (default).  
  - GROUP_ENABLED – Counts enabled agents only (exclusive with all).  
  - GROUP_MAX – Include the maximum number of agents during the game (can be combined with all or enabled). |
| type      | The AI object type for which to filter. Counts only the AI objects of the type specified. This parameter cannot be used with the GROUP_MAX flag. |

**GetGroupMember**

Returns entity that is at a specified index position in the specified group.

**Syntax**

```
AI.GetGroupMember(entityId|groupId, idx, flags, type)
```

Returns the script handler of the requested entity, or null if the requested index value is out of range.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>groupId</td>
</tr>
<tr>
<td>idx</td>
<td>Position in the index from 1 to n.</td>
</tr>
</tbody>
</table>
| flags      | A combination of one or more of the following flags:  
  - GROUP_ALL – Counts all agents in the group (default).  
  - GROUP_ENABLED – Counts enabled agents only (exclusive with all). |
| type        | The AI object type for which to filter. Returns only the AI objects of the type specified. This parameter cannot be used with the GROUP_MAX flag. |

**GetGroupOf**

Retrieves the group ID of a specified entity ID.

**Syntax**

```
AI.GetGroupOf(entityId)
```

Returns the group ID of the specified entity.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity whose group ID to return.</td>
</tr>
</tbody>
</table>

**GetGroupScopeUserCount**

**Syntax**
AI.GetGroupScopeUserCount(ScriptHandle entityIdHandle, const char* groupScopeName)

Returns the number of actors inside the group scope if greater than or equal to zero, or nil if an error occurs.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The entity ID of the agent for whom you want to access the group scope.</td>
</tr>
<tr>
<td>groupScopeName</td>
<td>The group scope name.</td>
</tr>
</tbody>
</table>

**GetGroupScriptTable**

**Syntax**

AI.GetGroupScriptTable(int groupID)

**GetGroupTarget**

Retrieves the most threatening attention target among the AI agents in a specified entity's group. See IAgent.h for a definition of alert status.

**Syntax**

AI.GetGroupTarget(entityId [,bHostileOnly [,bLiveOnly]])

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>Unique entity ID used to determine the group.</td>
</tr>
<tr>
<td>bHostileOnly (optional)</td>
<td>Flag indicating whether or not to include only hostile targets in group.</td>
</tr>
<tr>
<td>bLiveOnly (optional)</td>
<td>Flag indicating whether or not to include only live targets in group.</td>
</tr>
</tbody>
</table>

**GetGroupTargetCount**

Retrieves the number of attention targets among the AI agents in a specified entity's group.

**Syntax**

AI.GetGroupTargetCount(entityId [,bHostileOnly [,bLiveOnly]])

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>Unique entity ID used to determine the group.</td>
</tr>
<tr>
<td>bHostileOnly (optional)</td>
<td>Flag indicating whether or not to include only hostile targets in group.</td>
</tr>
</tbody>
</table>
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bLiveOnly (optional)</td>
<td>Flag indicating whether or not to include only live targets in group.</td>
</tr>
</tbody>
</table>

#### GetGroupTargetEntity

**Syntax**

```csharp
AI.GetGroupTargetEntity(int groupID)
```

#### GetGroupTargetThreat

**Syntax**

```csharp
AI.GetGroupTargetThreat(int groupID)
```

#### GetGroupTargetType

**Syntax**

```csharp
AI.GetGroupTargetType(int groupID)
```

#### GetLastUsedSmartObject

Retrieves the smart object last used by the user specified.

**Syntax**

```csharp
AI.GetLastUsedSmartObject(userEntityId)
```

Returns nil if there is no last used smart object or if an error has occurred; otherwise, returns the script table of the entity that was the smart object last used by the user specified.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userEntityId</td>
<td>The entity ID of the user to query for the last used smart object.</td>
</tr>
</tbody>
</table>

#### GetLeader

Gets the name of a specified group leader.

**Syntax**

```csharp
AI.GetLeader(groupID | entityID)
```

Returns the leader name.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupID</td>
<td>Unique group ID.</td>
</tr>
</tbody>
</table>
### GetMemoryFireType

**Syntax**

```cpp
AI.GetMemoryFireType(entityId)
```

Returns the method that the puppet uses for firing at its memory target.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

### GetNavigationType

Retrieves the navigation type at a specified entity's position.

**Syntax**

```cpp
AI.GetNavigationType(entityId)
```

Returns Navigation type, such as `NAV_TRIANGULAR`, `NAV_WAYPOINT_HUMAN`, `NAV_ROAD`, `NAV_VOLUME`, `NAV_WAYPOINT_3DSURFACE`, `NAV_FLIGHT`, `NAV_SMARTOBJECT`. See the `IAISystem::ENavigationType` definition for a complete list.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

### GetNearestEntitiesOfType

**Syntax**

```cpp
AI.GetNearestEntitiesOfType(entityId|objectname|position, AIObjectType, maxObjects, returnList [,objectFilter [,radius]])
```

Returns the number of found entities.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>Unique entity ID, AI object name, or position used to pinpoint the center position of the search.</td>
</tr>
<tr>
<td>objectname</td>
<td>Radius of the search area.</td>
</tr>
<tr>
<td>position.</td>
<td>AIObject type to search for (see <code>ScriptBindAI.cpp</code> and <code>Scripts/AIAccessor.lua</code> for a complete list of AIObject types).</td>
</tr>
</tbody>
</table>
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxObjects</td>
<td>Maximum number of objects to find.</td>
</tr>
<tr>
<td>return list</td>
<td>Lua table to hold the list of found entities (Lua handlers).</td>
</tr>
</tbody>
</table>
| (optional) objectFilter | A combination of one or more of the following search filter flags:  
- `AIOBJECTFILTER_SAMEFACTION` – Include only AI objects of the same species as the querying object.  
- `AIOBJECTFILTER_SAMEGROUP` – Include only AI objects of the same group as the querying object (or with no group).  
- `AIOBJECTFILTER_NOGROUP` – Include only AI objects with a Group ID of AI_NOGROUP.  
- `AIOBJECTFILTER_INCLUDEINACTIVE` – Include objects that are inactive. |

---

### GetNearestHidespot

Retrieves a specified entity's nearest hidepoint within a specified range.

**Syntax**

`AI.GetNearestHidespot(entityId, rangeMin, rangeMax [, center])`

Returns Point position, if found.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>rangeMin</td>
<td>Minimum range of search area.</td>
</tr>
<tr>
<td>rangeMax</td>
<td>Maximum range of search area</td>
</tr>
<tr>
<td>centre (optional)</td>
<td>Center point of the search area. If not specified, the entity's current position is used.</td>
</tr>
</tbody>
</table>

---

### GetNearestPathOfTypeInRange

Retrieves the type of path nearest to a specified point of interest for a specified entity. Paths use the same types as anchors and are specified in the path properties. The function only returns paths that match the entity's navigation caps. Navigation type is also specified in the path properties.

**Syntax**

`AI.GetNearestPathOfTypeInRange(entityId, pos, range, type [, devalue, useStartNode])`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>pos</td>
<td>Vector specifying to the point of interest.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>range</td>
<td>Search range.</td>
</tr>
<tr>
<td>type</td>
<td>Type of path to search for.</td>
</tr>
<tr>
<td>devalue (optional)</td>
<td>Time the returned path is marked as occupied.</td>
</tr>
<tr>
<td>useStartNode (optional)</td>
<td>Flag indicating whether or not to look a path with any point inside the range (useStartNode=0) or one with a start node inside the range (useStartNode=0).</td>
</tr>
</tbody>
</table>

**GetNearestPointOnPath**

Locates the point on a path nearest to a specified position.

**Syntax**

```lua
AI.GetNearestPointOnPath(entityId, pathname, vPos)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>pathname</td>
<td>Name of path.</td>
</tr>
<tr>
<td>vPos</td>
<td>Position to measure from.</td>
</tr>
</tbody>
</table>

**GetObjectBlackBoard**

Retrieves a specified object’s black board (a Lua table).

**Syntax**

```lua
AI.GetObjectBlackBoard(entityId)
```

Returns black board – if there is one; otherwise, nil.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId or entityName</td>
<td>An AI entity identifier.</td>
</tr>
</tbody>
</table>

**GetObjectRadius**

Retrieves the radius of a specified AI object.

**Syntax**

```lua
AI.GetObjectRadius(entityId)
```

Returns the radius size.
### GetParameter

Retrieves the value of an enumerated AI parameter for a specified entity.

**Syntax**

```plaintext
AI.GetParameter(entityId, paramEnum)
```

Returns the value of the parameter.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>paramEnum</td>
<td>The index of the parameter to get. See <code>AI.ChangeParameter()</code> for a complete list.</td>
</tr>
</tbody>
</table>

### GetPathLoop

**Syntax**

```plaintext
AI.GetPathLoop(entityId, pathname)
```

Returns true if path is successfully looped.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>pathname</td>
<td>Name of the path.</td>
</tr>
</tbody>
</table>

### GetPathSegNoOnPath

**Syntax**

```plaintext
AI.GetPathSegNoOnPath(entityId, pathname, vPos)
```

Returns Segment ratio (0.0 start point, 100.0 end point).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>pathname</td>
<td>Name of path.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>vPos</td>
<td>Position.</td>
</tr>
</tbody>
</table>

**GetPeakThreatLevel**

**Syntax**

```c
AI.GetPeakThreatLevel(ScriptHandle entityID)
```

**GetPeakThreatType**

**Syntax**

```c
AI.GetPeakThreatType(ScriptHandle entityID)
```

**GetPointOnPathBySegNo**

**Syntax**

```c
AI.GetPointOnPathBySegNo(entityId, pathname, segNo)
```

Returns Point by segment ratio (0.0 start point, 100.0 end point).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>pathname</td>
<td>Name of path.</td>
</tr>
<tr>
<td>segNo</td>
<td>Segment ratio.</td>
</tr>
</tbody>
</table>

**GetPosturePriority**

Sets the specified entity's posture priority.

**Syntax**

```c
AI.GetPosturePriority(ScriptHandle entityId, const char* postureName)
```

**GetPotentialTargetCount**

Retrieves the total number of a specified entity's potential targets.

**Syntax**

```c
AI.GetPotentialTargetCount(ScriptHandle entityID)
```
**GetPotentialTargetCountFromFaction**

Retrieves the number of an entity's potential targets that belong to a specified faction.

**Syntax**

```java
AI.GetPotentialTargetCountFromFaction(ScriptHandle entityID, const char* factionName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>name</td>
<td>Faction name.</td>
</tr>
</tbody>
</table>

**GetPredictedPosAlongPath**

Retrieves the predicted position of an AI agent along its path at a specified time.

**Syntax**

```java
AI.GetPredictedPosAlongPath(entityId, time, retPos)
```

Returns True if successful.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>time</td>
<td>Time (in seconds) to predict position.</td>
</tr>
<tr>
<td>retPos</td>
<td>Return point value of the predicted position.</td>
</tr>
</tbody>
</table>

**GetPreviousBehaviorName**

**Syntax**

```java
AI.GetPreviousBehaviorName(ScriptHandle entityID)
```

**GetPreviousPeakThreatLevel**

**Syntax**

```java
AI.GetPreviousPeakThreatLevel(ScriptHandle entityID)
```

**GetPreviousPeakThreatType**

**Syntax**

```java
AI.GetPreviousPeakThreatType(ScriptHandle entityID)
```
GetProbableTargetPosition

Retrieves the probable target position of a specified entity.

**Syntax**

```plaintext
AI.GetProbableTargetPosition(entityId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

GetRefPointDirection

Retrieves a specified entity's reference point direction.

**Syntax**

```plaintext
AI.GetRefPointDirection(entityId)
```

Returns a script vector (x,y,z) reference point direction.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

GetRefPointPosition

Retrieves a specified entity's reference point "world" position.

**Syntax**

```plaintext
AI.GetRefPointPosition(entityId)
```

Returns a script vector (x,y,z) reference point position.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

GetRefShapeName

Retrieves the name of a specified entity's reference shape.

**Syntax**

```plaintext
AI.GetRefShapeName(entityId)
```

Returns a reference shape name.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

**GetSoundPerceptionDescriptor**

Retrieves information about how the specified entity perceives sound types.

**Syntax**

```c
AI.GetSoundPerceptionDescriptor(entityId, soundType, descriptorTable)
```

Returns true if the information is successfully returned.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>Entity to get perception data on.</td>
</tr>
<tr>
<td>soundType</td>
<td>Type of sound stimulus to get data for.</td>
</tr>
<tr>
<td>descriptorTable</td>
<td>Location to store retrieved data.</td>
</tr>
</tbody>
</table>

**GetStance**

Retrieves the specified entity's stance.

**Syntax**

```c
AI.GetStance(entityId)
```

Returns entity stance (STANCE_*)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

**GetSubTypeOf**

Retrieves a specified entity's sub type.

**Syntax**

```c
AI.GetSubTypeOf(entityId)
```

Returns the entity sub type (as defined in IAgent.h).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>
GetTacticalPoints

Retrieves a point matching a description, related to a specified entity. Format of a point is: \{x,y,z\}.

Syntax

```cpp
AI.GetTacticalPoints(entityId, tacPointSpec, point)
```

Returns true if a valid point is found; otherwise, false.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>AI's entity.</td>
</tr>
<tr>
<td>tacPointSpec.</td>
<td>A table specifying the points required.</td>
</tr>
<tr>
<td>point</td>
<td>Coordinates of the point found.</td>
</tr>
</tbody>
</table>

GetTargetSubType

Retrieves the subtype of a specified entity's current attention target.

Syntax

```cpp
AI.GetTargetSubType(entityId)
```

Returns an attention target subtype. See `IAgent.h` for a list of target type definitions.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

GetTargetType

Retrieves the type of a specified entity's current attention target.

Syntax

```cpp
AI.GetTargetType(entityId)
```

Returns an attention target type, such as AITARGET_NONE, AITARGET_MEMORY, AITARGET_BEACON, AITARGET_ENEMY. See `IAgent.h` for a list of target type definitions.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

GetTotalLengthOfPath

Retrieves total length of the specified path.

Syntax
AI.GetTotalLengthOfPath(entityId, pathname)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>pathname</td>
<td>Name of path.</td>
</tr>
</tbody>
</table>

**GetTypeOf**

Retrieves a specified entity's type.

**Syntax**

AI.GetTypeOf(entityId)

Returns the entity type (as defined in IAgent.h).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

**GetUnitCount**

Retrieves the number of units the leader knows about. The leader is identified based on the group ID of the entity.

**Syntax**

AI.GetUnitCount(entityId, unitProperties)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>unitProperties</td>
<td>Binary mask of unit properties type for which the attack is requested, in the following form:</td>
</tr>
<tr>
<td></td>
<td><code>UPR_* + UPR* (UPR_COMBAT_GROUND + UPR_COMBAT_FLIGHT)</code></td>
</tr>
<tr>
<td></td>
<td>See IAgent.h for a definition of unit properties <code>UPR_*</code>.</td>
</tr>
</tbody>
</table>

**GetUnitInRank**

Retrieves the entity that holds the specified rank position in the specified group.

**Syntax**

AI.GetUnitInRank(groupID [,rank])

Returns entity script table of the ranked unit.
### ScriptBind Engine Functions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupID</td>
<td>The ID of the group that contains the entity to retrieve.</td>
</tr>
<tr>
<td>rank</td>
<td>The rank position of the entity to retrieve. If null or a value less than or equal to zero is specified, retrieves the entity with the highest rank in the entity. The value of the highest rank is 1.</td>
</tr>
</tbody>
</table>

---

**GoTo**

Allows the specified entity to move to a certain destination.

**Syntax**

```
AI.GoTo(entityId, vDestination)
```

---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>AI's entity.</td>
</tr>
<tr>
<td>vDestination</td>
<td>.</td>
</tr>
</tbody>
</table>

---

**Hostile**

Determines whether or not two entities are hostile.

**Syntax**

```
AI.Hostile(entityId, entity2Id|AIObjectName)
```

Returns true if the entities are hostile.

---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>ID of the first AI entity.</td>
</tr>
<tr>
<td>entity2Id</td>
<td>ID of the second AI entity, or AIObject name.</td>
</tr>
</tbody>
</table>

---

**IgnoreCurrentHideObject**

Marks the current hide object as unreachable; it will be omitted from future hidespot selections.

**Syntax**

```
AI.IgnoreCurrentHideObject(entityId)
```

---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>
IntersectsForbidden
Determines whether or not the specified line is in a forbidden region.

Syntax

AI.IntersectsForbidden(Vec3 start, Vec3 end)

Returns intersected position or end (if there is no intersection).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>Vector in format {x,y,z}.</td>
</tr>
<tr>
<td>end</td>
<td>Vector in format {x,y,z}.</td>
</tr>
</tbody>
</table>

IsAgentInAgentFOV
Determines whether or not one entity is in the field of view of another entity.

Syntax

AI.IsAgentInAgentFOV(entityId, entityId2)

Returns the first value true if the agent is within the entity FOV; the second value true if the agent is within the entity’s primary FOV, or false if the agent is within the entity’s secondary FOV.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The AI entity whose FOV to check.</td>
</tr>
<tr>
<td>entityId2</td>
<td>The entity ID of the agent.</td>
</tr>
</tbody>
</table>

IsAgentInTargetFOV
Determines whether or not the entity is in the FOV of the attention target.

Syntax

AI.IsAgentInTargetFOV(entityId, fov)

Returns true if in the FOV of the attention target; otherwise, false.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>fov</td>
<td>FOV of the enemy in degrees.</td>
</tr>
</tbody>
</table>

IsAimReady

Syntax
AI.IsAimReady(ScriptHandle entityIdHandle)

**IsCoverCompromised**

**Syntax**

```cpp
AI.IsCoverCompromised(entityId)
```

Returns true if the cover has been compromised; otherwise, nil.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>AI's entity.</td>
</tr>
</tbody>
</table>

**IsEnabled**

Checks that the entity is AI-enabled.

**Syntax**

```cpp
AI.IsEnabled(entityId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

**IsFireEnabled**

Determines whether or not the AI is allowed to fire or not.

**Syntax**

```cpp
AI.IsFireEnabled(entityId)
```

Returns true if AI is enabled to fire

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

**IsInCover**

Determines whether or not the agent is using cover.

**Syntax**

```cpp
AI.IsInCover(entityId)
```
IsLowHealthPauseActive

Syntax

AI.IsLowHealthPauseActive(ScriptHandle entityID)

IsLowOnAmmo

Syntax

AI.IsLowOnAmmo(entityId)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>AI’s entity.</td>
</tr>
<tr>
<td>threshold</td>
<td>The ammo percentage threshold.</td>
</tr>
</tbody>
</table>

IsMoving

Determines whether or not the agent wants to move.

Syntax

AI.IsMoving(entityId)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

IsMovingInCover

Syntax

AI.IsMovingInCover(entityId)

Returns true – Agent is moving in cover nil – if not

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>AI’s entity.</td>
</tr>
</tbody>
</table>

IsMovingToCover

Determines whether or not the agent is running to cover.

Syntax

AI.IsMovingToCover(entityId)
### IsOutOfAmmo

**Syntax**

``` whispers
AI.IsOutOfAmmo(entityId)
```

Returns true if the specified entity is out of ammunition; otherwise, nil.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>AI's entity.</td>
</tr>
</tbody>
</table>

### IsPersonallyHostile

**Syntax**

``` whispers
AI.IsPersonallyHostile(ScriptHandle entityID, ScriptHandle hostileID)
```

### IsPointInFlightRegion

Determines whether or not a specified point is in the flight region.

**Syntax**

``` whispers
AI.IsPointInFlightRegion(point)
```

Returns true if the point is in the flight region.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>point</td>
<td>Vector in format {x,y,z}.</td>
</tr>
</tbody>
</table>

### IsPointInsideGenericShape

Determines whether or not a point is inside a specified shape.

**Syntax**

``` whispers
AI.IsPointInsideGenericShape(position, shapeName[, checkHeight])
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>Position to check.</td>
</tr>
</tbody>
</table>
Lumberyard Legacy Reference
ScriptBind Engine Functions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shapeName</td>
<td>Name of the shape to test (returned by AI.GetEnclosingGenericShapeOfType).</td>
</tr>
<tr>
<td>checkHeight</td>
<td>Flag indicating whether or not to test for shape height. (default=false). If set to true, the test will check the space between shape.aabb.min.z and shape.aabb.min.z+shape.height.</td>
</tr>
</tbody>
</table>

**IsPointInWaterRegion**

Determines whether or not the point is in the water region.

**Syntax**

```c
AI.IsPointInWaterRegion(point)
```

Returns a value that indicates water or ground level. Values greater than 0 mean there is water.

**IsPunchableObjectValid**

Determines whether or not a punchable object is valid.

**Syntax**

```c
AI.IsPunchableObjectValid(userId, objectId, origPos)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userId</td>
<td>User ID.</td>
</tr>
<tr>
<td>objectId</td>
<td>Object ID.</td>
</tr>
<tr>
<td>origPos</td>
<td>Object position in the world.</td>
</tr>
</tbody>
</table>

**IsTakingCover**

**Syntax**

```c
AI.IsTakingCover(entityId, [distanceThreshold])
```

Returns true if the specified agent is either in cover or running to cover; otherwise, nil.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>AI's entity.</td>
</tr>
<tr>
<td>distanceThreshold</td>
<td>(Optional) Distance over which an agent that is running to cover is considered to not yet have taken cover.</td>
</tr>
</tbody>
</table>

**LoadBehaviors**

**Syntax**

```c
800
```
AI.LoadBehaviors(const char* folderName, const char* extension)

**LogComment**

Writes additional information to the log for debugging purposes.

**Syntax**

```
AI.LogComment(szMessage)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>szMessage</td>
<td>The message to write to the log.</td>
</tr>
</tbody>
</table>

**LogEvent**

Writes event-driven information to the log that for debugging purposes. Events may occur on a per-frame or a per AI update basis.

**Syntax**

```
AI.LogEvent(szMessage)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>szMessage</td>
<td>The message to write to the log.</td>
</tr>
</tbody>
</table>

**LogProgress**

Writes progress messages to the log.

**Syntax**

```
AI.LogProgress(szMessage)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>szMessage</td>
<td>The message to write to the log.</td>
</tr>
</tbody>
</table>

**MeleePunchableObject**

**Syntax**

```
AI.MeleePunchableObject(entityId, objectId, origPos)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the AI entity.</td>
</tr>
</tbody>
</table>
**ModifySmartObjectStates**

Adds or removes smart object states for a specified entity.

**Syntax**

```cpp
AI.ModifySmartObjectStates(entityId, listStates)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>listStates</td>
<td>The list of state names to be added or removed (such as “Closed, Locked”,</td>
</tr>
<tr>
<td></td>
<td>“Open, Unlocked, Busy”).</td>
</tr>
</tbody>
</table>

**ParseTables**

**Syntax**

```cpp
AI.ParseTables(int firstTable, bool parseMovementAbility, IFunctionHandler* pH, AIObjectParams& aiParams, bool& updateAlways)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstTable</td>
<td>Properties table.</td>
</tr>
<tr>
<td>parseMovementAbility</td>
<td>True to parse movement ability, false otherwise.</td>
</tr>
<tr>
<td>aiParams</td>
<td>AI parameters.</td>
</tr>
<tr>
<td>updateAlways</td>
<td>True to always update; false otherwise.</td>
</tr>
</tbody>
</table>

**PlayCommunication**

Plays communication on the AI agent.

**Syntax**

```cpp
AI.PlayCommunication(ScriptHandle entityId, const char* commName, const char* channelName, float contextExpiry)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>
### PlayReadabilitySound

Plays readability sound on the AI agent. This call does not do any filtering like playing readability using signals.

**Syntax**

```c
AI.PlayReadabilitySound(entityId, soundName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>soundName</td>
<td>The name of the readability sound signal to play.</td>
</tr>
<tr>
<td>stopPreviousSounds</td>
<td>True if any currently playing readability should be stopped in favor of this one.</td>
</tr>
<tr>
<td>responseDelayMin</td>
<td>Minimum (or exact, if no maximum) delay for the response readability to play.</td>
</tr>
<tr>
<td>responseDelayMax</td>
<td>Maximum delay for the response readability to play.</td>
</tr>
</tbody>
</table>

### ProcessBalancedDamage

Processes balanced damage.

**Syntax**

```c
AI.ProcessBalancedDamage(pShooterEntity, pTargetEntity, damage, damageType)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pShooterEntity</td>
<td>Shooter ID.</td>
</tr>
<tr>
<td>pTargetEntity</td>
<td>Target ID.</td>
</tr>
<tr>
<td>damage</td>
<td>Hit damage.</td>
</tr>
<tr>
<td>damageType</td>
<td>Hit damage type.</td>
</tr>
</tbody>
</table>

### QueueBubbleMessage

**Syntax**

```c
AI.QueueBubbleMessage(ScriptHandle entityID, const char* message)
```
RecComment

Records a comment with AI Debug Recorder. For information about the AI Debug Recorder, see Using the AI Debug Recorder.

Syntax

```
AI.RecComment(szMessage)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>szMessage</td>
<td>Message line to be displayed in Recorder view.</td>
</tr>
</tbody>
</table>

RegisterDamageRegion

Registers a spherical region that causes damage (so should be avoided in pathfinding). The owner entity position is used as the region center. The function can be called multiple times to update the region position.

Syntax

```
AI.RegisterDamageRegion(entityId, radius)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>radius</td>
<td>The radius of the spherical region. If less than or equal to zero, the region is disabled.</td>
</tr>
</tbody>
</table>

RegisterInterestedActor

Registers the interested actor with the interest system. Any errors go to the error log.

Syntax

```
AI.RegisterInterestedActor(ScriptHandle entityId, float fInterestFilter, float fAngleInDegrees)
```

Returns true if a valid update was performed; otherwise, nil. Nil can be returned if the interest system is disabled or the parameters are not valid.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the AI entity.</td>
</tr>
</tbody>
</table>

RegisterInterestingEntity

Registers the specified entity with the interest system. Any errors go to the error log.

Syntax
AI.RegisterInterestingEntity(ScriptHandle entityId, float radius, float baseInterest, const char* actionName, Vec3 offset, float pause, int shared)

Returns true if a valid update was performed; otherwise, nil. Nil can be returned if the interest system is disabled or the parameters are not valid.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

RegisterTacticalPointQuery

Retrieves a query ID for the specified tactical point query.

Syntax

AI.RegisterTacticalPointQuery(querySpecTable)

Returns > 0 – If the query was parsed successfully 0 – Otherwise

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>querySpecTable</td>
<td>Table specifying the query. For more information, see AI Tactical Point System.</td>
</tr>
</tbody>
</table>

RegisterTargetTrack

Registers an AI entity to use a specified target track configuration for target selection. The parameter ai_TargetTracking must be set to '2'.

Syntax

AI.RegisterTargetTrack(entityId, configuration, targetLimit, classThreat)

Returns true if successfully registered.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>configuration</td>
<td>Target track configuration.</td>
</tr>
<tr>
<td>targetLimit</td>
<td>The number of agents who can target the AI at any specified time (0 for infinite).</td>
</tr>
<tr>
<td>classThreat (optional)</td>
<td>Initial class threat value.</td>
</tr>
</tbody>
</table>

RemovePersonallyHostile

Syntax
AI.RemovePersonallyHostile(ScriptHandle entityID, ScriptHandle hostileID)

**RequestAttack**

In a group with a leader, allows the leader to issue a request for a group attack behavior against the enemy. After this request, the CLeader may create an attack leader action (CLeaderAction_Attack_*).

**Syntax**

```lua
AI.RequestAttack(entityId, unitProperties, attackTypeList [,duration])
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>Unique entity ID used to determine the group leader.</td>
</tr>
</tbody>
</table>
| unitProperties | Binary mask of unit properties type for which the attack is requested, in the following form:  
                     UPR_* + UPR* (UPR_COMBAT_GROUND + UPR_COMBAT_FLIGHT)  
                     See IAgent.h for a definition of unit properties UPR_*.
| attackTypeList | Lua table containing a prioritized list of preferred attack strategies (leader action subtypes). The list must be in the following format:  
                     {LAS_*, LAS_*,...} (LAS_ATTACK_ROW,LAS_ATTACK_FLANK)  
                     which means that the first attempt will be an Attack_row action, and if that fails an attack_flank.  
                     See IAgent.h for a definition of LeaderActionSubtype (LAS_*) action types. |
| duration (optional) | Maximum duration in seconds (default = 0). |

**RequestToStopMovement**

**Syntax**

```lua
AI.RequestToStopMovement(ScriptHandle entityId)
```

**ResetAgentLookAtPos**

Resets the specified entity's previous call to AgentLookAtPos().

**Syntax**

```lua
AI.ResetAgentLookAtPos(entityId)
```
### ResetAgentState

Resets a particular aspect of the agent's state, such as "lean".

**Syntax**

```c
AI.ResetAgentState(ScriptHandle entityId, const char * stateLabel)
```

Returns nil

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the AI entity.</td>
</tr>
<tr>
<td>stateLabel</td>
<td>String describing the state that must be reset to default.</td>
</tr>
</tbody>
</table>

### ResetParameters

Resets all parameters for a specified entity.

**Syntax**

```c
AI.ResetParameters(entityId, bProcessMovement, PropertiesTable, PropertiesInstanceTable)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity whose parameters you want to reset.</td>
</tr>
<tr>
<td>bProcessMovement</td>
<td>True to reset movement data; otherwise, false.</td>
</tr>
<tr>
<td>PropertiesTable</td>
<td>The Lua table that contains the entity's properties.</td>
</tr>
<tr>
<td>PropertiesInstanceTable</td>
<td>The Lua table that contains instance-specific entity properties.</td>
</tr>
</tbody>
</table>

### ResetPersonallyHostiles

**Syntax**

```c
AI.ResetPersonallyHostiles(ScriptHandle entityID, ScriptHandle hostileID)
```

### ScaleFormation

Changes the scale factor of a specified entity's formation (if one exists).

**Syntax**

```c
AI.ScaleFormation(entityId, scale)
```

Returns true if formation scaling was successful.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>scale</td>
<td>Scale factor.</td>
</tr>
</tbody>
</table>

**SequenceBehaviorReady**

**Syntax**

```c
AI.SequenceBehaviorReady(ScriptHandle entityId)
```

**SequenceInterruptibleBehaviorLeft**

**Syntax**

```c
AI.SequenceInterruptibleBehaviorLeft(ScriptHandle entityId)
```

**SequenceNonInterruptibleBehaviorLeft**

**Syntax**

```c
AI.SequenceNonInterruptibleBehaviorLeft(ScriptHandle entityId)
```

**SetAlarmed**

Sets the entity to be "perception alarmed".

**Syntax**

```c
AI.SetAlarmed(entityId)
```

**SetAnimationTag**

Sets a mannequin animation tag.

**Syntax**

```c
AI.SetAnimationTag(ScriptHandle entityID, const char* tagName)
```

Returns a default result code (in Lua: void).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the AI entity on which to set the animation tag.</td>
</tr>
<tr>
<td>tagName</td>
<td>The name of the animation tag that should be set (case insensitive).</td>
</tr>
</tbody>
</table>

**SetAssessmentMultiplier**

Sets the assessment multiplier factor for the specified AIOBJECT type.

**Syntax**

```c
808
```
**AI.SetAssesmentMultiplier(AIObjectType, multiplier)**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIObjectType</td>
<td>Type of AIObject. See ScriptBindAI.cpp for a complete list of AIObject types.</td>
</tr>
<tr>
<td>multiplier</td>
<td>Assessment multiplier factor.</td>
</tr>
</tbody>
</table>

**SetAttentionTarget**

Sets a new attention target.

**Syntax**

```plaintext
AI.SetAttentionTarget(entityId, targetId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>targetId</td>
<td>Target's entity ID.</td>
</tr>
</tbody>
</table>

**SetBeaconPosition**

Sets the beacon's position for the specified entity/object's group.

**Syntax**

```plaintext
AI.SetBeaconPosition(entityId | AIObjectName, pos)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>Unique entity ID or AI object name.</td>
</tr>
<tr>
<td>pos</td>
<td>Vector ({x,y,z}) where the beacon position will be set.</td>
</tr>
</tbody>
</table>

**SetBehaviorTreeEvaluationEnabled**

**Syntax**

```plaintext
AI.SetBehaviorTreeEvaluationEnabled(ScriptHandle entityID, bool enable)
```

**SetBehaviorVariable**

Sets a behaviour variable for the specified actor.

**Syntax**

```plaintext
```
AI.SetBehaviorVariable(ScriptHandle entityId, const char* variableName, bool value)

**SetCollisionAvoidanceRadiusIncrement**

**Syntax**

AI.SetCollisionAvoidanceRadiusIncrement(ScriptHandle entityId, float radius)

**SetContinuousMotion**

**Syntax**

AI.SetContinuousMotion(ScriptHandle entityID, bool continuousMotion)

**SetCoverCompromised**

**Syntax**

AI.SetCoverCompromised(entityId)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the AI entity.</td>
</tr>
</tbody>
</table>

**SetEntitySpeedRange**

**Syntax**

AI.SetEntitySpeedRange(userEntityId, urgency, defaultSpeed, minSpeed, maxSpeed, stance = all)

Returns true if the operation was successful and false otherwise

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userEntityId</td>
<td>The entity ID of the user for which its last used smart object is needed.</td>
</tr>
<tr>
<td>urgency</td>
<td>The integer value specifying the movement urgency (see AgentMovementSpeeds::EAgentMovementUrgency).</td>
</tr>
<tr>
<td>defaultSpeed</td>
<td>Floating point value that specifies the default speed.</td>
</tr>
<tr>
<td>minSpeed</td>
<td>Floating point value that specifies the minimum speed.</td>
</tr>
</tbody>
</table>

**SetExtraPriority**

Sets a extra priority value to the specified enemy entity.

**Syntax**

AI.SetExtraPriority(enemyEntityId, increment)
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enemyEntityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>float increment</td>
<td>Value to add to the target’s priority.</td>
</tr>
</tbody>
</table>

**SetFactionOf**

Sets the faction to which the specified entity belongs.

**Syntax**

```
AI.SetFactionOf(ScriptHandle entityId, const char* factionName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity whose faction to return.</td>
</tr>
<tr>
<td>factionName</td>
<td>The name of the faction to assign to the specified entity.</td>
</tr>
</tbody>
</table>

**SetFactionThreatMultiplier**

Sets the threat multiplier factor for the specified species. A return value of 0 indicates that the species is not hostile to any other species.

**Syntax**

```
AI.SetFactionThreatMultiplier(nSpecies, multiplier)
```

**SetFireMode**

Sets fire mode immediately.

**Syntax**

```
AI.SetFireMode(entityId, mode)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>firemode</td>
<td>New fire mode.</td>
</tr>
</tbody>
</table>

**SetFormationAngleThreshold**

Sets the relative position inside the formation.

**Syntax**

```
AI.SetFormationAngleThreshold(entityId, fAngleThreshold)
```
### SetFormationLookingPoint

Sets the relative looking point position inside the formation.

**Syntax**

```plaintext
AI.SetFormationLookingPoint(entityId, v3RelativePosition)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the AI entity.</td>
</tr>
<tr>
<td>v3RelativePosition</td>
<td>Table with format {x,y,z} storing the new relative looking point.</td>
</tr>
</tbody>
</table>

### SetFormationPosition

Sets the relative position inside the formation.

**Syntax**

```plaintext
AI.SetFormationPosition(entityId, v2RelativePosition)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the AI entity.</td>
</tr>
<tr>
<td>v2RelativePosition</td>
<td>Table with format {x,y} storing the new relative position.</td>
</tr>
</tbody>
</table>

### SetFormationUpdate

Sets the update flag for a specified entity's formation (if one exists). If this flag is false, the formation is no longer updated.

**Syntax**

```plaintext
AI.SetFormationUpdate(entityId, update)
```

Returns true if the request was successful.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>update</td>
<td>True to update the flag; otherwise, false.</td>
</tr>
</tbody>
</table>
SetFormationUpdateSight

Sets a random angle rotation for a specified entity's formation sight directions.

Syntax

```
AI.SetFormationUpdateSight(entityId, range, minTime, maxTime)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>range</td>
<td>Angle of rotation (0,360) around the default sight direction.</td>
</tr>
<tr>
<td>minTime (optional)</td>
<td>Minimum timespan for changing the direction (default = 2).</td>
</tr>
<tr>
<td>maxTime (optional)</td>
<td>Minimum timespan for changing the direction (default = minTime).</td>
</tr>
</tbody>
</table>

SetIgnorant

Sets the specified AI entity to ignore system signals, visual stimuli and sound stimuli.

Syntax

```
AI.SetIgnorant(entityId, ignorant)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the AI entity.</td>
</tr>
<tr>
<td>ignorant</td>
<td>A flag indicating whether or not the entity ignores system signals. 0 specifies do not ignore; 1 specifies ignore.</td>
</tr>
</tbody>
</table>

SetInCover

Syntax

```
AI.SetInCover(entityId, bool inCover)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the AI entity.</td>
</tr>
<tr>
<td>inCover</td>
<td>Specifies whether the entity should be set to be in cover or not.</td>
</tr>
</tbody>
</table>

SetLeader

Sets a specified entity as the group leader. This action associates a CLeader object with the entity, creating it if one doesn't exist. Only one leader can be set per group.

Syntax
AI.SetLeader(entityID)

Returns true if successful.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityID</td>
<td>Unique entity ID to set as leader.</td>
</tr>
</tbody>
</table>

**SetMemoryFireType**

Sets how the AI agent handles firing at its memory target.

**Syntax**

```
AI.SetMemoryFireType(entityId, type)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>type</td>
<td>Memory fire type. Possible values from enum EMemoryFireType in IAgent.h:</td>
</tr>
<tr>
<td></td>
<td>eMFT_Disabled = 0, // Never allowed to fire at memory</td>
</tr>
<tr>
<td></td>
<td>eMFT_UseCoverFireTime, // Can fire at memory using the</td>
</tr>
<tr>
<td></td>
<td>weapon's cover fire time</td>
</tr>
<tr>
<td></td>
<td>eMFT_Always, // Always allowed to fire at memory</td>
</tr>
</tbody>
</table>

**SetMovementContext**

Sets the specified entity's movement context.

**Syntax**

```
AI.SetMovementContext(ScriptHandle entityId, int context)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>context</td>
<td>context value.</td>
</tr>
</tbody>
</table>

**SetPathAttributeToFollow**

Sets the attribute of a specified entity's path.

**Syntax**

```
AI.SetPathAttributeToFollow(entityId, flag)
```
### SetPathToFollow
Sets the path for a specified entity to follow.

**Syntax**

```c
AI.SetPathToFollow(entityId, pathName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>pathName</td>
<td>Name of the path to be followed.</td>
</tr>
</tbody>
</table>

### SetPFBlockerRadius

**Syntax**

```c
AI.SetPFBlockerRadius(entityId, blocker, radius)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>blocker</td>
<td>Name of the path to be followed.</td>
</tr>
<tr>
<td>radius</td>
<td>Name of the path to be followed.</td>
</tr>
</tbody>
</table>

### SetPointListToFollow
Sets a point list for a specified entity's path.

**Syntax**

```c
AI.SetPointListToFollow(entityId, pointlist, howmanypoints, bspline [, navtype])
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>pointList</td>
<td>List of points for the entity to follow, expressed as a set of local vectors:{{x=0.0, y=0.0, z=0.0},.</td>
</tr>
<tr>
<td>howmanypoints</td>
<td>Number of points in the list.</td>
</tr>
<tr>
<td>bspline</td>
<td>Flag indicating whether or not the path line is recalculated using spline interpolation.</td>
</tr>
</tbody>
</table>
### ScriptBind Engine Functions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>navtype</td>
<td>(Optional) Navigation type (default = IAISystem::NAV_FLIGHT).</td>
</tr>
</tbody>
</table>

**SetPosturePriority**

Sets the specified entity's posture priority.

**Syntax**

```cpp
AI.SetPosturePriority(ScriptHandle entityId, const char* postureName, float priority)
```

**SetPostures**

Sets the specified entity's postures.

**Syntax**

```cpp
AI.SetPostures(ScriptHandle entityId, SmartScriptTable postures)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>postures</td>
<td>The table of postures.</td>
</tr>
</tbody>
</table>

**SetRefPointAtDefensePos**

Sets a specified entity's reference point position to an intermediate distance between the entity's attention target and a specified point.

**Syntax**

```cpp
AI.SetRefPointAtDefensePos(entityId, point2defend, distance)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>point2defend</td>
<td>Point to defend.</td>
</tr>
<tr>
<td>distance</td>
<td>Maximum distance between reference point and point to defend.</td>
</tr>
</tbody>
</table>

**SetRefPointDirection**

Sets a specified entity's reference point direction.

**Syntax**

```cpp
AI.SetRefPointDirection(vRefPointDir)
```
### SetRefPointPosition

Sets a specified entity's reference point "world" position.

**Syntax**

```plaintext
AI.SetRefPointPosition(entityId, vRefPointPos)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>vRefPointPos</td>
<td>World position as a (script)vector (x,y,z) value.</td>
</tr>
</tbody>
</table>

### SetRefPointRadius

Sets a specified entity's reference point radius.

**Syntax**

```plaintext
AI.SetRefPointRadius(entityId, radius)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>radius</td>
<td>The reference point radius.</td>
</tr>
</tbody>
</table>

### SetRefpointToAnchor

Sets a reference point to an anchor.

**Syntax**

```plaintext
AI.SetRefpointToAnchor(entityId, rangeMin, rangeMax, findType, findMethod)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the AI entity.</td>
</tr>
<tr>
<td>rangeMin</td>
<td>Minimum range.</td>
</tr>
<tr>
<td>rangeMax</td>
<td>Maximum range.</td>
</tr>
<tr>
<td>findType</td>
<td>Finding type.</td>
</tr>
<tr>
<td>findMethod</td>
<td>Finding method.</td>
</tr>
</tbody>
</table>
### SetRefpointToPunchableObject
Sets the reference point to the punchable object.

**Syntax**
AI.SetRefpointToPunchableObject(entityId, range)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the AI entity.</td>
</tr>
<tr>
<td>range</td>
<td>Range for the punchable object.</td>
</tr>
</tbody>
</table>

### SetRefShapeName
Sets the name of a specified entity's reference shape.

**Syntax**
AI.SetRefShapeName(entityId, name)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the reference shape.</td>
</tr>
</tbody>
</table>

### SetSmartObjectState
Sets a single smart object state, replacing all other states.

**Syntax**
AI.SetSmartObjectState(entityId, stateName)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the new state to set for the smart object (such as &quot;Idle&quot;).</td>
</tr>
</tbody>
</table>

### SetSoundPerceptionDescriptor
Sets information about how the specified entity perceives sound types.

**Syntax**
AI.SetSoundPerceptionDescriptor(entityId, soundType, descriptorTable)
Returns True if information successfully saved.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>Entity to set perception data for.</td>
</tr>
<tr>
<td>soundType</td>
<td>Type of sound stimulus to set data for.</td>
</tr>
<tr>
<td>descriptorTable</td>
<td>Perception data to saved.</td>
</tr>
</tbody>
</table>

**SetSpeed**
Sets the entity's current speed (urgency).

**Syntax**

```
AI.SetSpeed(entityId, urgency)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>AI's entity.</td>
</tr>
<tr>
<td>urgency</td>
<td>A float value that specifies the movement urgency (see AgentMovementSpeeds::EAgentMovementUrgency).</td>
</tr>
</tbody>
</table>

**SetStance**
Sets the specified entity's stance.

**Syntax**

```
AI.SetStance(entityId, stance)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>stance</td>
<td>The stance value (STANCE_*)</td>
</tr>
</tbody>
</table>

**SetTargetTrackClassThreat**
Sets the class threat for a specified entity's target track.

**Syntax**

```
AI.SetTargetTrackClassThreat(entityId, classThreat)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>
### SetTempTargetPriority

Sets a specified entity’s selection priority for a temporary target over other potential targets.

**Syntax**

```c
AI.SetTempTargetPriority(entityId, priority)
```

Returns True if successfully updated.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>priority</td>
<td>New priority value.</td>
</tr>
</tbody>
</table>

### SetTerritoryShapeName

Sets the territory shape of the specified AI entity.

**Syntax**

```c
AI.SetTerritoryShapeName(entityId, shapeName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>shapeName</td>
<td>Name of the shape to set.</td>
</tr>
</tbody>
</table>

### SetUnitProperties

Sets the leader’s knowledge about the unit’s combat capabilities. The leader is identified based on the group ID of the entity.

**Syntax**

```c
AI.SetUnitProperties(entityId, unitProperties)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>
| unitProperties | Binary mask of unit properties in the following form:  
`UPR_*` + `UPR*` (`UPR_COMBAT_GROUND` + `UPR_COMBAT_FLIGHT`)  
See `IAgent.h` for a definition of the `UPR_*` unit properties. |
SetUseSecondaryVehicleWeapon

Enables or disables the AI object's ability to use the secondary weapon when firing from a vehicle gunner seat if possible.

Syntax

```plaintext
AI.SetUseSecondaryVehicleWeapon(entityId, bUseSecondary)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>bUseSecondary</td>
<td>Specify true to use the secondary weapon; otherwise, false.</td>
</tr>
</tbody>
</table>

Signal

Adds a signal to the sender's signal queue even if another signal with same text is present.

Syntax

```plaintext
AI.Signal(signalFilter, signalType, signalText, senderId [, signalExtraData])
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>signalFilter</td>
<td>The signal filter.</td>
</tr>
<tr>
<td>signalType</td>
<td>The signal type.</td>
</tr>
<tr>
<td>signalText</td>
<td>Signal text that is processed by the receivers, either in a Lua callback with the same name as the text or directly by the CAIObject.</td>
</tr>
<tr>
<td>senderId</td>
<td>The ID of the sender.</td>
</tr>
<tr>
<td>signalExtraData</td>
<td>Optional. A Lua table containing additional data. It can contain the following data types:</td>
</tr>
<tr>
<td></td>
<td>• point – A vector in the format {x,y,z}.</td>
</tr>
<tr>
<td></td>
<td>• point2 – A vector in the format {x,y,z}.</td>
</tr>
<tr>
<td></td>
<td>• ObjectName – A string.</td>
</tr>
<tr>
<td></td>
<td>• id – An entity ID.</td>
</tr>
<tr>
<td></td>
<td>• fValue – A float value.</td>
</tr>
<tr>
<td></td>
<td>• iValue – An integer value.</td>
</tr>
<tr>
<td></td>
<td>• iValue2 – A second integer value.</td>
</tr>
</tbody>
</table>

SmartObjectEvent

Executes a smart action.

Syntax

```plaintext
AI.SmartObjectEvent(actionName, userEntityId, objectEntityId [, vRefPoint])
```
Returns 0 if a smart object rule was not found or if a non-zero ID was inserted to execute the action.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionName</td>
<td>The name of the smart action.</td>
</tr>
<tr>
<td>usedEntityId</td>
<td>The entity ID of the user who wants to execute the smart action, or none if</td>
</tr>
<tr>
<td></td>
<td>the user is unknown.</td>
</tr>
<tr>
<td>objectEntityId</td>
<td>The entity ID of the object on which the smart action is to be executed, or</td>
</tr>
<tr>
<td></td>
<td>none if the object is unknown.</td>
</tr>
<tr>
<td>vRefPoint</td>
<td>Optional. The reference point to be used instead of the user's attention</td>
</tr>
<tr>
<td></td>
<td>target position.</td>
</tr>
</tbody>
</table>

**SoundEvent**

Generates a sound event with the specified parameters in the AI system.

**Syntax**

```plaintext
AI.SoundEvent(position, radius, threat, interest, entityId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>Origin of the sound event.</td>
</tr>
<tr>
<td>radius</td>
<td>Area the sound event is heard in.</td>
</tr>
<tr>
<td>threat</td>
<td>Sound event property.</td>
</tr>
<tr>
<td>interest</td>
<td>Sound event property.</td>
</tr>
<tr>
<td>entityId</td>
<td>Unique entity ID that generates the sound event.</td>
</tr>
</tbody>
</table>

**StopCommunication**

Stops specified communication.

**Syntax**

```plaintext
AI.StopCommunication(ScriptHandle playID)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>playID</td>
<td>The ID of the communication to stop.</td>
</tr>
</tbody>
</table>

**ThrowGrenade**

Throws a specified grenade at a target type without interrupting the fire mode.

**Syntax**
AI.ThrowGrenade(entityId, grenadeType, regTargetType)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>grenadeType</td>
<td>Requested grenade type (see ERequestedGrenadeType).</td>
</tr>
<tr>
<td>regTargetType</td>
<td>The grenade target type (see AI_REG_*).</td>
</tr>
</tbody>
</table>

**UnregisterInterestedActor**

Unregisters the entity with the interest system. Any errors are recorded in the error log.

**Syntax**

AI.UnregisterInterestedActor(ScriptHandle entityId)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

**UnregisterInterestingEntity**

Unregisters the specified entity with the interest system. Any errors are recorded in the error log.

**Syntax**

AI.UnregisterInterestingEntity(ScriptHandle entityId)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>

**UnregisterTargetTrack**

Unregisters an AI object from the target track manager. The parameter ai_TargetTracking must be set to '2'.

**Syntax**

AI.UnregisterTargetTrack(entityId)

Returns true if successfully unregistered.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
</tbody>
</table>
UpdateGlobalPerceptionScale

Syntax

AI.UpdateGlobalPerceptionScale(float visualScale, float audioScale)

UpdateTempTarget

Updates the position of the specified entity's temporary potential target.

Syntax

AI.UpdateTempTarget(entityId, vPos)

Returns true if successfully updated.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>vPos</td>
<td>New position of the temporary target.</td>
</tr>
</tbody>
</table>

UpTargetPriority

Changes a specified entity's target priority value for a specified target, if the target is on the entity's target list.

Syntax

AI.UpTargetPriority(entityId, targetId, increment)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity.</td>
</tr>
<tr>
<td>targetId</td>
<td>The entity ID of the target.</td>
</tr>
<tr>
<td>increment</td>
<td>New value for the target priority.</td>
</tr>
</tbody>
</table>

VisualEvent

Generates a visual event with the specified parameters in the AI system.

Syntax

AI.VisualEvent(entityId, targetId)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The ID of the entity that receives the visual event.</td>
</tr>
</tbody>
</table>
### ScriptBind Engine Functions

#### Parameter | Description
--- | ---
| targetId | The ID of the visual target (that the entity is seeing).

**Warning**

Writes a warning message to the log about data or script errors.

**Syntax**

```
AI.Warning(szMessage)
```

| Parameter | Description |
--- | ---|
| szMessage | The message to write to the log.

**ScriptBind_Entity**

CryLua is deprecated and will be removed in a future version of Lumberyard.

Lists the entity related Lua script bind functions.

#### Activate

Activates or deactivates the entity. `Activate` ignores the update policy and forces an entity to activate or deactivate. All active entities are updated every frame.

**Warning**

Having too many active entities can affect performance.

**Syntax**

```
Entity.Activate(int bActive)
```

**Parameters**

| Parameter | Description |
--- | ---|
| bActive | Specify true to make the entity active; false to make it inactive.

#### ActivateOutput

**Syntax**

```
Entity.ActivateOutput()
```

#### ActivatePlayerPhysics

**Syntax**
AddConstraint

Syntax

Entity.AddConstraint()

AddImpulse

Apply an impulse to the entity. At least four parameters need to be provided for a linear impulse. For an additional angular impulse, at least seven parameters need to be provided.

Syntax

Entity.AddImpulse(ipart, position, direction, linearImpulse, linearImpulseScale, angularAxis, angularImpulse, massScale)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipart</td>
<td>The index of the part that receives the impulse.</td>
</tr>
<tr>
<td>position</td>
<td>The point (in world coordinates) where the impulse is applied. Set this to (0, 0, 0) to ignore it.</td>
</tr>
<tr>
<td>direction</td>
<td>The direction in which the impulse is applied.</td>
</tr>
<tr>
<td>linearImpulse</td>
<td>The force of the linear impulse.</td>
</tr>
<tr>
<td>linearImpulseScale</td>
<td>Scaling of the linear impulse. (Default: 1.0)</td>
</tr>
<tr>
<td>angularAxis</td>
<td>The axis on which the angular impulse is applied.</td>
</tr>
<tr>
<td>angularImpulse</td>
<td>The force of the angular impulse.</td>
</tr>
<tr>
<td>massScale</td>
<td>Mass scaling of the angular impulse. (Default: 1.0)</td>
</tr>
</tbody>
</table>

AttachChild

Syntax

Entity.AttachChild(ScriptHandle childEntityId, int flags)

AttachSurfaceEffect

Syntax

Entity.AttachSurfaceEffect(int nSlot, const char *effect, bool countPerUnit, const char *form, const char *typ, float countScale, float sizeScale)

AuxAudioProxiesMoveWithEntity

Set whether AuxAudioProxies should move with the entity or not.
Syntax

Entity.AuxAudioProxiesMoveWithEntity(bool const bCanMoveWithEntity)

Returns: nil

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bCanMoveWithEntity</td>
<td>Boolean parameter to enable or disable</td>
</tr>
</tbody>
</table>

AwakeCharacterPhysics

Syntax

Entity.AwakeCharacterPhysics(int nSlot, const char *sRootBoneName, int nAwake)

AwakeEnvironment

Syntax

Entity.AwakeEnvironment()

AwakePhysics

Syntax

Entity.AwakePhysics(int nAwake)

BreakToPieces

Breaks static geometry in slot 0 into sub objects and spawn them as particles or entities.

Syntax

Entity.BreakToPieces(int nSlot, int nPiecesSlot, float fExplodeImp, Vec3 vHitPt, Vec3 vHitImp, float fLifeTime, bool bSurfaceEffects)

CalcWorldAnglesFromRelativeDir

Syntax

Entity.CalcWorldAnglesFromRelativeDir(Vec3 dir)

CancelarSubpipe

Syntax

Entity.CancelSubpipe()
**ChangeAttachmentMaterial**

**Syntax**

```c
Entity.ChangeAttachmentMaterial(const char *attachmentName, const char *materialName)
```

**CharacterUpdateAlways**

**Syntax**

```c
Entity.CharacterUpdateAlways(int characterSlot, bool updateAlways)
```

**CharacterUpdateOnRender**

**Syntax**

```c
Entity.CharacterUpdateOnRender(int characterSlot, bool bUpdateOnRender)
```

**CheckCollisions**

**Syntax**

```c
Entity.CheckCollisions()
```

**CheckShaderParamCallbacks**

Check all the currently set shader param callbacks on the renderproxy with the current state of the entity.

**Syntax**

```c
Entity.UpdateShaderParamCallback()
```

**CloneMaterial**

Replace material on the slot with a cloned version of the material. Cloned material can be freely changed uniquely for this entity.

**Syntax**

```c
Entity.CloneMaterial(int slot)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slot</td>
<td>ID of the slot on which to clone material.</td>
</tr>
<tr>
<td>sSubMaterialName</td>
<td>If this is a non empty string this specific sub-material is cloned.</td>
</tr>
</tbody>
</table>

**CopySlotTM**

Copies the TM (Transformation Matrix) of the slot.
Syntax

Entity.CopySlotTM(int destSlot, int srcSlot, bool includeTranslation)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destSlot</td>
<td>Destination slot identifier.</td>
</tr>
<tr>
<td>srcSlot</td>
<td>Source slot identifier.</td>
</tr>
<tr>
<td>includeTranslation</td>
<td>True to include the translation, false otherwise.</td>
</tr>
</tbody>
</table>

CountLinks

Counts all outgoing links of the entity.

Syntax

Entity.CountLinks()

Returns: Number of outgoing links.

CreateAuxAudioProxy

Creates an additional AudioProxy managed by the EntityAudioProxy. The created AuxAudioProxy will move and rotate with the parent EntityAudioProxy.

Syntax

Entity.CreateAuxAudioProxy()

Returns: Returns the ID of the additionally created AudioProxy.

CreateBoneAttachment

Syntax

Entity.CreateBoneAttachment(int characterSlot, const char *boneName, const char *attachmentName)

CreateCameraComponent

Create a camera component for the entity. Allows the entity to serve as camera source for material assigned to the entity.

Syntax

Entity.CreateCameraComponent()

CreateDRSProxy

Creates a Dynamic Response System Proxy
**Entity.CreateDRSProxy()**

*Returns:* Returns the ID of the created proxy.

**CreateLink**

Creates a new outgoing link for this entity.

**Syntax**

`Entity.CreateLink(const char *name)`

*Returns:* nothing

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the link. Does not have to be unique among all the links of this entity. Multiple links with the same name can coexist.</td>
</tr>
<tr>
<td>(optional) targetId</td>
<td>If specified, the ID of the entity the link shall target. If not specified or 0 then the link will not target anything. Default value: 0</td>
</tr>
</tbody>
</table>

**CreateRenderComponent**

Create a render component object for the entity. Allows an entity to be rendered immediately without loading any assets.

**Syntax**

`Entity.CreateRenderComponent()`

**CreateSkinAttachment**

**Syntax**

`Entity.CreateSkinAttachment(int characterSlot, const char *attachmentName)`

**Damage**

**Syntax**

`Entity.Damage()`

**DeleteParticleEmitter**

Deletes particles emitter from 3dengine.

**Syntax**

`Entity.DeleteParticleEmitter()`
Entity.DeleteParticleEmitter(int slot)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slot</td>
<td>slot number</td>
</tr>
</tbody>
</table>

DeleteThis

Deletes the current entity.

Syntax

Entity.DeleteThis()

DestroyAttachment

Syntax

Entity.DestroyAttachment(int characterSlot, const char *attachmentName)

DestroyPhysics

Syntax

Entity.DestroyPhysics()

DetachAll

Syntax

Entity.DetachAll()

DetachThis

Syntax

Entity.DetachThis()

DisableAnimationEvent

Syntax

Entity.DisableAnimationEvent(int nSlot, const char *sAnimation)

DrawSlot

Enables/Disables drawing of object or character at specified slot of the entity.

Syntax
Entity.DrawSlot(int nSlot, int nEnable)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>Slot identifier.</td>
</tr>
<tr>
<td>nEnable</td>
<td>1-Enable drawing, 0-Disable drawing.</td>
</tr>
</tbody>
</table>

EnableBoneAnimation

**Syntax**

Entity.EnableBoneAnimation(int characterSlot, int layer, const char *boneName, bool status)

EnableBoneAnimationAll

**Syntax**

Entity.EnableBoneAnimationAll(int characterSlot, int layer, bool status)

EnableDecals

Enables decals.

**Syntax**

Entity.EnableDecals(int slot, bool enable)

EnableInheritXForm

Enables/Disable entity from inheriting transformation from the parent.

**Syntax**

Entity.EnableInheritXForm(bool bEnable)

EnableMaterialLayer

**Syntax**

Entity.EnableMaterialLayer(bool enable, int layer)

EnablePhysics

**Syntax**

Entity.EnablePhysics(bool bEnable)

EnableProceduralFacialAnimation

**Syntax**
Entity.EnableProceduralFacialAnimation(bool enable)

**ExecuteAudioTrigger**

Execute the specified audio trigger and attach it to the entity. The created audio object will move and rotate with the entity.

**Syntax**

Entity.ExecuteAudioTrigger(ScriptHandle const hTriggerID, ScriptHandle const hAudioProxyLocalID)

**Returns:** nil

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTriggerID</td>
<td>the audio trigger ID handle</td>
</tr>
<tr>
<td>hAudioProxyLocalID</td>
<td>The ID of the AuxAudioProxy that is local to the EntityAudioProxy. To address the default AuxAudioProxy, pass 1. To address all AuxAudioProxy instances, pass 0.</td>
</tr>
</tbody>
</table>

**FadeGlobalDensity**

Sets the fade global density.

**Syntax**

Entity.FadeGlobalDensity(int nSlot, float fadeTime, float newGlobalDensity)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>nSlot identifier.</td>
</tr>
<tr>
<td>fadeTime</td>
<td>.</td>
</tr>
<tr>
<td>newGlobalDensity</td>
<td>.</td>
</tr>
</tbody>
</table>

**ForceCharacterUpdate**

**Syntax**

Entity.ForceCharacterUpdate(int characterSlot, bool updateAlways)

**ForwardTriggerEventsTo**

**Syntax**

Entity.ForceCharacterUpdate(ScriptHandle entityID)
**FreeAllSlots**
Delete all objects on every slot part of the entity.

**Syntax**

```
Entity.FreeAllSlots()
```

**FreeSlot**
Delete all objects from specified slot.

**Syntax**

```
Entity.FreeSlot(int nSlot)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>Slot identifier.</td>
</tr>
</tbody>
</table>

**GetAIName**

**Syntax**

```
Entity.GetAIName()
```

**GetAllAuxAudioProxiesID**
Returns the ID used to address all AuxAudioProxy of the parent EntityAudioProxy.

**Syntax**

```
Entity.GetAllAuxAudioProxiesID()
```

**Returns**: Returns the ID used to address all AuxAudioProxy of the parent EntityAudioProxy.

**GetAngles**
Gets the angle of the entity.

**Syntax**

```
Entity.GetAngles()
```

**GetAnimationLength**

**Syntax**

```
Entity.GetAnimationLength(int characterSlot, const char *animation)
```

**GetAnimationTime**

**Syntax**

```
```
GetAnimationTime

Syntax

Entity.GetAnimationTime(int nSlot, int nLayer)

GetArchetype

Retrieve the archetype of the entity.

Syntax

Entity.GetArchetype()

Returns: name of entity archetype, nil if no archetype.

GetAttachmentBone

Syntax

Entity.GetAttachmentBone(int characterSlot, const char *attachmentName)

GetAttachmentCGF

Syntax

Entity.GetAttachmentCGF(int characterSlot, const char *attachmentName)

GetBoneAngularVelocity

Syntax

Entity.GetBoneAngularVelocity(int characterSlot, const char *boneName)

GetBoneDir

Syntax

Entity.GetBoneDir()

GetBoneLocal

Syntax

Entity.GetBoneLocal(const char *boneName, Vec3 trgDir)

GetBoneNameFromTable

Syntax

Entity.GetBoneNameFromTable()

GetBonePos

Syntax

Entity.GetBonePos()
**GetBoneVelocity**

Syntax

```
Entity.GetBoneVelocity(int characterSlot, const char *boneName)
```

**GetCenterOfMassPos**

Gets the position of the entity center of mass.

Syntax

```
Entity.GetCenterOfMassPos()
```

**GetCharacter**

Gets the character for the specified slot if there is any.

Syntax

```
Entity.GetCharacter(int nSlot)
```

**GetChild**

Syntax

```
Entity.GetChild(int nIndex)
```

**GetChildCount**

Syntax

```
Entity.GetChildCount()
```

**GetCurAnimation**

Syntax

```
Entity.GetCurAnimation()
```

**GetDefaultAuxAudioProxyID**

Returns the ID of the default AudioProxy of the parent EntityAudioProxy.

Syntax

```
Entity.GetDefaultAuxAudioProxyID()
```

**Returns:** Returns the ID of the default AudioProxy of the parent EntityAudioProxy.

**GetDirectionVector**

Syntax
Entity.GetDirectionVector()

**GetDistance**

**Syntax**

```c
float Entity.GetDistance(entityId)
```

**Returns:** The distance from entity specified with entityId/

**GetEntitiesInContact**

**Syntax**

```c
Entity.GetEntitiesInContact()
```

**GetEntityMaterial**

**Syntax**

```c
Entity.GetEntityMaterial()
```

**GetExplosionImpulse**

**Syntax**

```c
Entity.GetExplosionImpulse()
```

**GetExplosionObstruction**

**Syntax**

```c
Entity.GetExplosionObstruction()
```

**GetFlags**

**Syntax**

```c
Entity.GetFlags()
```

**GetFlagsExtended**

**Syntax**

```c
Entity.GetFlagsExtended()
```

**GetGeomCachePrecachedTime**

Gets time delta from current playback position to last ready to play frame.

**Syntax**

```c
837
```
GetGeomCachePrecachedTime()

GetGravity

Syntax

Entity.GetGravity()

GetHelperAngles

Syntax

Entity.GetHelperAngles()

GetHelperDir

Syntax

Entity.GetHelperDir()

GetHelperPos

Syntax

Entity.GetHelperPos()

GetLink

Returns the link at given index.

Syntax

Entity.GetLink()

Returns: The script table of the entity that the i'th link is targeting or nil if the specified index is out of bounds.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ith</td>
<td>The index of the link that shall be returned.</td>
</tr>
</tbody>
</table>

GetLinkName

Returns the name of the link that is targeting the entity with given ID.

Syntax

Entity.GetLinkName(ScriptHandle targetId)

Returns: The name of the i'th link targeting given entity or nil if no such link exists.
### GetLinkTarget

Returns the ID of the entity that given link is targeting.

**Syntax**

```
Entity.GetLinkTarget(const char *name)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the link.</td>
</tr>
<tr>
<td>(optional) ith</td>
<td>If specified, the i'th link with given name for which to look up the targeted entity. Default value: 0 (first link with given name)</td>
</tr>
</tbody>
</table>

### GetLocalAngles

**Syntax**

```
Vec3 Entity.GetLocalAngles(vAngles)
```

### GetLocalBBox

**Syntax**

```
Entity.GetLocalBBox()
```

### GetLocalPos

**Syntax**

```
Vec3 Entity.GetLocalPos()
```

### GetLocalScale

**Syntax**

```
float Entity.GetLocalScale()
```
GetLodRatio
Syntax
Entity.GetLodRatio()

GetMass
Syntax
Entity.GetMass()

GetMaterial
Syntax
Entity.GetMaterial()

GetMaterialFloat
Change material parameter.
Syntax
Entity.GetMaterialFloat(int slot,int nSubMtlId,const char *sParamName)

Returns: Material parameter value.

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slot</td>
<td>ID of the slot on which slot to change material.</td>
</tr>
<tr>
<td>nSubMtlId</td>
<td>Specify submaterial by Id.</td>
</tr>
<tr>
<td>sParamName</td>
<td>Name of the material parameter.</td>
</tr>
</tbody>
</table>

GetMaterialVec3
Syntax
Entity.GetMaterialVec3(int slot,int nSubMtlId,const char *sParamName)

GetName
Syntax
Entity.GetName()

GetParent
Syntax
GetParentSlot

Syntax

Entity.GetParentSlot(int child)

GetPhysicalStats

Some more physics related.

Syntax

Entity.GetPhysicalStats()

GetPos

Gets the position of the entity.

Syntax

Entity.GetPos()

GetProjectedWorldBBox

Syntax

Entity.GetProjectedWorldBBox()

GetRawId

Returns entityId in raw numeric format.

Syntax

Entity.GetRawId()

GetScale

Gets the scaling value for the entity.

Syntax

Entity.GetScale()

GetSlotAngles

Gets the slot angles.

Syntax

Entity.GetSlotAngles(int nSlot)
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>nSlot identifier.</td>
</tr>
</tbody>
</table>

**GetSlotCount**

Gets the count of the slots.

**Syntax**

```cpp
Entity.GetSlotCount()
```

**GetSlotHelperPos**

**Syntax**

```cpp
Entity.GetSlotHelperPos(int slot, const char *helperName, bool objectSpace)
```

**GetSlotPos**

Gets the slot position.

**Syntax**

```cpp
Entity.GetSlotPos(int nSlot)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>nSlot identifier.</td>
</tr>
</tbody>
</table>

**GetSlotScale**

Gets the slot scale amount.

**Syntax**

```cpp
Entity.GetSlotScale(int nSlot)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>nSlot identifier.</td>
</tr>
</tbody>
</table>

**GetSlotWorldDir**

Gets the World direction of the slot.

**Syntax**
Entity.GetSlotWorldDir(int nSlot)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>Slot identifier.</td>
</tr>
</tbody>
</table>

**GetSlotWorldPos**

Gets the World position of the slot.

**Syntax**

Entity.GetSlotWorldPos(int nSlot)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>Slot identifier.</td>
</tr>
</tbody>
</table>

**GetSpeed**

**Syntax**

Entity.GetSpeed()

**GetState**

**Syntax**

Entity.GetState()

**GetSubmergedVolume**

**Syntax**

Entity.GetSubmergedVolume(int slot, Vec3 planeNormal, Vec3 planeOrigin)

**GetTimeOfDayHour**

**Syntax**

Entity.GetTimeOfDayHour()

**Returns**: current time of day as a float value.

**GetTimeSinceLastSeen**

**Syntax**

Entity.GetTimeSinceLastSeen()
**GetTouchedPoint**

Retrieves point of collision for rigid body.

**Syntax**

```c
Entity.GetTouchedPoint()
```

**GetTouchedSurfaceID**

**Syntax**

```c
Entity.GetTouchedSurfaceID()
```

**GetTriggerBBox**

**Syntax**

```c
Entity.GetTriggerBBox()
```

**GetUpdateRadius**

**Syntax**

```c
Entity.GetUpdateRadius()
```

**GetVelocity**

**Syntax**

```c
Entity.GetVelocity()
```

**GetVelocityEx**

**Syntax**

```c
Entity.GetVelocityEx()
```

**GetViewDistanceMultiplier**

Get the view distance multiplier.

**Syntax**

```c
Entity.GetViewDistanceMultiplier()
```

**GetVolume**

**Syntax**

```c
Entity.GetVolume(int slot)
```
GetWorldAngles
Syntax
Vec3 Entity.GetWorldAngles(vAngles)

GetWorldBBox
Syntax
Entity.GetWorldBBox()

GetWorldBoundsCenter
Gets the world bbox center for the entity (defaults to entity position if no bbox present).
Syntax
Entity.GetWorldBoundsCenter()

GetWorldDir
Syntax
Vec3 Entity.GetWorldDir()

GetWorldPos
Syntax
Vec3 Entity.GetWorldPos()

GetWorldScale
Syntax
float Entity.GetWorldScale()

GotoState
Syntax
Entity.GotoState(const char *sState)

HasFlags
Syntax
Entity.HasFlags(int flags)

HasFlagsExtended
Syntax
Entity.HasFlagsExtended(int flags)

**Hide**

**Syntax**

Entity.Hide()

**HideAllAttachments**

**Syntax**

Entity.HideAllAttachments(int characterSlot, bool hide, bool hideShadow)

**HideAttachment**

**Syntax**

Entity.HideAttachment(int characterSlot, const char *attachmentName, bool hide, bool hideShadow)

**HideAttachmentMaster**

**Syntax**

Entity.HideAttachmentMaster(int characterSlot, bool hide)

**IgnorePhysicsUpdatesOnSlot**

Ignore physics when updating the position of a slot.

**Syntax**

Entity.IgnorePhysicsUpdatesOnSlot(int nSlot)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>Slot identifier.</td>
</tr>
</tbody>
</table>

**InsertSubpipe**

**Syntax**

Entity.InsertSubpipe()

**IntersectRay**

**Syntax**

Entity.IntersectRay(int slot, Vec3 rayOrigin, Vec3 rayDir, float maxDistance)
**InvalidateTrigger**

**Syntax**

```plaintext
Entity.InvalidateTrigger()
```

**IsActive**

Retrieve active status of entity.

**Syntax**

```plaintext
Entity.IsActive(bActivate)
```

**Returns:** true - Entity is active. false - Entity is not active.

**IsAnimationRunning**

**Syntax**

```plaintext
Entity.IsAnimationRunning(int characterSlot, int layer)
```

**Returns:** nil or not nil

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>characterSlot</td>
<td>Index of the character slot.</td>
</tr>
<tr>
<td>layer</td>
<td>Index of the animation layer.</td>
</tr>
</tbody>
</table>

**IsColliding**

**Syntax**

```plaintext
Entity.IsColliding()
```

**IsEntityInside**

**Syntax**

```plaintext
float Entity.IsEntityInside(entityId)
```

**IsEntityInsideArea**

**Syntax**

```plaintext
Entity.IsEntityInsideArea(int areaId, ScriptHandle entityId)
```

**IsFromPool**

Returns if the entity came from an entity pool.
Entity.IsFromPool()

**Returns:** true - Entity is from a pool. (Bookmarked) false - Entity is not from a pool. (Not bookmarked)

**IsGeomCacheStreaming**

Syntax

Entity.IsGeomCacheStreaming()

**Returns:** true if geom cache is streaming.

**IsHidden**

Syntax

Entity.IsHidden()

**IsInState**

Syntax

Entity.IsInState(const char *sState)

**IsPointInsideArea**

Syntax

Entity.IsPointInsideArea(int areaId, Vec3 point)

**IsSlotCharacter**

Checks if the slot is a character.

Syntax

Entity.IsSlotCharacter(int slot)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slot</td>
<td>Slot identifier.</td>
</tr>
</tbody>
</table>

**IsSlotGeometry**

Checks if the slot is a geometry.

Syntax

Entity.IsSlotGeometry(int slot)
**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slot</td>
<td>Slot identifier.</td>
</tr>
</tbody>
</table>

**IsSlotLight**

Checks if the slot is a light.

**Syntax**

```
Entity.IsSlotLight(int slot)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slot</td>
<td>Slot identifier.</td>
</tr>
</tbody>
</table>

**IsSlotParticleEmitter**

Checks if the slot is a particle emitter.

**Syntax**

```
Entity.IsSlotParticleEmitter(int slot)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slot</td>
<td>Slot identifier.</td>
</tr>
</tbody>
</table>

**IsSlotValid**

Checks if the slot is valid.

**Syntax**

```
Entity.IsSlotValid(int nSlot)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>Slot identifier.</td>
</tr>
</tbody>
</table>

**IsUsingPipe**

**Syntax**

```
Entity.IsUsingPipe()
```
**Returns**: True if the entity is running a goalpipe or has it inserted; otherwise, false.

**KillTimer**

**Syntax**

```
Entity.KillTimer()
```

**LoadCharacter**

Load CGF geometry into the entity slot.

**Syntax**

```
Entity.LoadCharacter(int nSlot, const char *sFilename)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>Slot identifier.</td>
</tr>
<tr>
<td>sFilename</td>
<td>CGF geometry file name.</td>
</tr>
</tbody>
</table>

**LoadCloud**

Loads the cloud XML file into the entity slot.

**Syntax**

```
Entity.LoadCloud(int nSlot, const char *sFilename)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>Slot identifier.</td>
</tr>
<tr>
<td>sFilename</td>
<td>Filename.</td>
</tr>
</tbody>
</table>

**LoadFogVolume**

Loads the fog volume XML file into the entity slot.

**Syntax**

```
Entity.LoadFogVolume(int nSlot, SmartScriptTable table)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>Slot identifier.</td>
</tr>
<tr>
<td>table</td>
<td>Table with fog volume properties.</td>
</tr>
</tbody>
</table>
LoadGeomCache
Load geom cache into the entity slot.

Syntax

```c
Entity.LoadGeomCache(int nSlot, const char *sFilename)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>Slot identifier.</td>
</tr>
<tr>
<td>sFilename</td>
<td>CAX file name.</td>
</tr>
</tbody>
</table>

LoadLight
Load CGF geometry into the entity slot.

Syntax

```c
Entity.LoadLight(int nSlot, SmartScriptTable table)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>Slot identifier.</td>
</tr>
<tr>
<td>table</td>
<td>Table with all the light information.</td>
</tr>
</tbody>
</table>

LoadObject
Load CGF geometry into the entity slot.

Syntax

```c
Entity.LoadObject(int nSlot, const char *sFilename)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>Slot identifier.</td>
</tr>
<tr>
<td>sFilename</td>
<td>CGF geometry file name.</td>
</tr>
</tbody>
</table>

LoadObjectLattice
Load lattice into the entity slot.

Syntax
Entity.LoadObjectLattice(int nSlot)

**LoadObjectWithFlags**
Load CGF geometry into the entity slot.

**Syntax**

Entity.LoadObjectWithFlags(int nSlot, const char *sFilename, const int nFlags)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>Slot identifier.</td>
</tr>
<tr>
<td>sFilename</td>
<td>CGF geometry file name.</td>
</tr>
<tr>
<td>nFlags</td>
<td>entity load flags</td>
</tr>
</tbody>
</table>

**LoadParticleEffect**
Load CGF geometry into the entity slot.

**Syntax**

Entity.LoadParticleEffect(int nSlot, const char *sEffectName, SmartScriptTable table)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>Slot identifier.</td>
</tr>
<tr>
<td>sEffectName</td>
<td>Name of the particle effect (Ex: &quot;explosions/rocket&quot;).</td>
</tr>
<tr>
<td>(optional) bPrime</td>
<td>Whether effect starts fully primed to equilibrium state.</td>
</tr>
<tr>
<td>(optional) fPulsePeriod</td>
<td>Time period between particle effect restarts.</td>
</tr>
<tr>
<td>(optional) fScale</td>
<td>Size scale to apply to particles</td>
</tr>
<tr>
<td>(optional) fCountScale</td>
<td>Count multiplier to apply to particles</td>
</tr>
<tr>
<td>(optional) bScalePerUnit</td>
<td>Scale size by attachment extent</td>
</tr>
<tr>
<td>(optional) bCountPerUnit</td>
<td>Scale count by attachment extent</td>
</tr>
<tr>
<td>(optional) sAttachType</td>
<td>string for EGeomType</td>
</tr>
<tr>
<td>(optional) sAttachForm</td>
<td>string for EGeomForm</td>
</tr>
</tbody>
</table>

**LoadSubObject**
Load geometry of one CGF node into the entity slot.
Syntax

```c
Entity.LoadSubObject(int nSlot, const char *sFilename, const char *sGeomName)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>Slot identifier.</td>
</tr>
<tr>
<td>sFilename</td>
<td>CGF geometry file name.</td>
</tr>
<tr>
<td>sGeomName</td>
<td>Name of the node inside CGF geometry.</td>
</tr>
</tbody>
</table>

**LoadVolumeObject**

Loads volume object.

Syntax

```c
Entity.LoadVolumeObject(int nSlot, const char* sFilename)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>Slot identifier.</td>
</tr>
<tr>
<td>sFilename</td>
<td>File name of the volume object.</td>
</tr>
</tbody>
</table>

**LookAt**

Orient the entity to look at a world space position.

Syntax

```c
Entity.LookAt(Vec3 target, Vec3 axis, float angle)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>The position to look at.</td>
</tr>
<tr>
<td>axis</td>
<td>The correction axis. The quat type is not supported.</td>
</tr>
<tr>
<td>angle</td>
<td>The correction angle in radians. The quat type is not supported.</td>
</tr>
</tbody>
</table>

**MultiplyWithSlotTM**

Multiplies with the TM (Transformation Matrix) of the slot.

Syntax
Entity.MultiplyWithSlotTM(int slot, Vec3 pos)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slot</td>
<td>Slot identifier.</td>
</tr>
<tr>
<td>pos</td>
<td>Position vector.</td>
</tr>
</tbody>
</table>

**NetPresent**

**Syntax**

Entity.NetPresent()

**NoBulletForce**

**Syntax**

Entity.NoBulletForce(bool state)

**NoExplosionCollision**

**Syntax**

Entity.NoExplosionCollision()

**PassParamsToPipe**

**Syntax**

Entity.PassParamsToPipe()

**Physicalize**

Create physical entity from the specified entity slot.

**Syntax**

Entity.Physicalize(int nSlot, int nPhysicsType, SmartScriptTable physicsParams)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>Slot identifier of the entity to physicalize. Specify -1 to use geometries from all slots.</td>
</tr>
<tr>
<td>nPhysicsType</td>
<td>Type of physical entity to create. For possible values, see the nPhysicsType Keys table later in this section.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>physicsParams</td>
<td>Table with physicalization parameters. For more information, see the <a href="#">physicsParams Table Keys</a> table later in this section.</td>
</tr>
</tbody>
</table>

### nPhysicsType Keys

<table>
<thead>
<tr>
<th>Physics Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE_AREA</td>
<td>Physical Area (Sphere,Box,Geometry or Shape).</td>
</tr>
<tr>
<td>PE_ARTICULATED</td>
<td>Ragdolls or other articulated physical entities that consist of rigid bodies connected by joints.</td>
</tr>
<tr>
<td>PE_LIVING</td>
<td>Live physical entity that can move through the physical world and interact with it.</td>
</tr>
<tr>
<td>PE_NONE</td>
<td>No physics.</td>
</tr>
<tr>
<td>PE_PARTICLE</td>
<td>A physical particle entity that it has only mass and radius.</td>
</tr>
<tr>
<td>PE_RIGID</td>
<td>Rigid body physical entity. Can have infinite mass (specified by setting mass to 0).</td>
</tr>
<tr>
<td>PE_ROPE</td>
<td>A physical representation of a rope. The rope can hang freely or connect two physical entities.</td>
</tr>
<tr>
<td>PE_SOFT</td>
<td>A system of non-rigidly connected vertices that can interact with the environment. Used for soft body physics like cloth simulation.</td>
</tr>
<tr>
<td>PE_STATIC</td>
<td>A static, immovable physical entity.</td>
</tr>
<tr>
<td>PE_WHEELEDVEHICLE</td>
<td>Physical vehicle with wheels.</td>
</tr>
</tbody>
</table>

**Note**
For more information about physical entity types, see [Physical Entities](#).

### physicsParams Table Keys

<table>
<thead>
<tr>
<th>Physics Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>area</td>
<td>This table must be set when Physics Type is PE_AREA. For more information, see the <a href="#">Area Table Keys</a> table later in this section.</td>
</tr>
<tr>
<td>density</td>
<td>Object density, only used if mass is not specified or -1.</td>
</tr>
<tr>
<td>flags</td>
<td>Physical entity flags.</td>
</tr>
<tr>
<td>living</td>
<td>This table must be set when Physics Type is PE_LIVING. For more information, see the <a href="#">Living Table Keys</a> table later in this section.</td>
</tr>
<tr>
<td>mass</td>
<td>Object mass, only used if density is not specified or -1.</td>
</tr>
<tr>
<td>particle</td>
<td>This table must be set when Physics Type is PE_PARTICLE. For more information, see the <a href="#">Particle Table Keys</a> table later in this section.</td>
</tr>
<tr>
<td>partid</td>
<td>Index of the articulated body part to which the new physical entity will be attached.</td>
</tr>
<tr>
<td>stiffness</td>
<td>Scale of character joint stiffness (multiplied with stiffness values specified from the exported model).</td>
</tr>
</tbody>
</table>
Particle Table Keys

<table>
<thead>
<tr>
<th>Particle Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accel_lift</td>
<td>Acceleration that lifts particle with the current speed</td>
</tr>
<tr>
<td>accel_thrust</td>
<td>Acceleration along direction of movement</td>
</tr>
<tr>
<td>air_resistance</td>
<td>The air resistance coefficient, ( F = kv )</td>
</tr>
<tr>
<td>constant_orientation</td>
<td>(0,1) Keep constant orientation</td>
</tr>
<tr>
<td>gravity</td>
<td>Gravity force vector to the air</td>
</tr>
<tr>
<td>mass</td>
<td>Particle mass</td>
</tr>
<tr>
<td>min_bounce_vel</td>
<td>Minimal velocity at which particle bounces off the surface</td>
</tr>
<tr>
<td>no_path_alignment</td>
<td>(0,1) Do not align particle orientation to the movement path</td>
</tr>
<tr>
<td>no_roll</td>
<td>(0,1) Do not roll particle on terrain</td>
</tr>
<tr>
<td>no_spin</td>
<td>(0,1) Do not spin particle in air</td>
</tr>
<tr>
<td>radius</td>
<td>Particle pseudo radius</td>
</tr>
<tr>
<td>single_contact</td>
<td>(0,1) Calculate only one first contact</td>
</tr>
<tr>
<td>thickness</td>
<td>Thickness when lying on a surface (if 0, the radius is used)</td>
</tr>
<tr>
<td>velocity</td>
<td>Velocity direction and magnitude vector</td>
</tr>
<tr>
<td>water_gravity</td>
<td>Gravity force vector when in the water.</td>
</tr>
<tr>
<td>water_resistance</td>
<td>Water resistance coefficient, ( F = kv )</td>
</tr>
</tbody>
</table>

Living Table Keys

<table>
<thead>
<tr>
<th>Living Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>air_resistance</td>
<td>Air control coefficient 0..1, 1 - special value (total control of movement)</td>
</tr>
<tr>
<td>gravity</td>
<td>Vertical gravity magnitude</td>
</tr>
<tr>
<td>head_radius</td>
<td>Radius of the head</td>
</tr>
<tr>
<td>height</td>
<td>Vertical offset of collision geometry center</td>
</tr>
<tr>
<td>height_eye</td>
<td>Vertical offset of the camera</td>
</tr>
<tr>
<td>height_head</td>
<td>Vertical offset of the head</td>
</tr>
<tr>
<td>height_pivot</td>
<td>Offset from central ground position that is considered the entity center</td>
</tr>
<tr>
<td>inertia</td>
<td>Inertia coefficient, the greater the value, the less the inertia; 0 means no inertia.</td>
</tr>
<tr>
<td>mass</td>
<td>Mass of the player (in kg)</td>
</tr>
<tr>
<td>max_climb_angle</td>
<td>Player cannot climb surface which slope is steeper than this angle (in radians)</td>
</tr>
<tr>
<td>max_jump_angle</td>
<td>Player is not allowed to jump towards ground if this angle is exceeded (in radians)</td>
</tr>
</tbody>
</table>
### Living Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_vel_ground</td>
<td>Player cannot stand on surfaces that are moving faster than this (in radians)</td>
</tr>
<tr>
<td>min_fall_angle</td>
<td>Player starts falling when slope is steeper than this (in radians)</td>
</tr>
<tr>
<td>min_slide_angle</td>
<td>If surface slope is more than this angle, player starts sliding (in radians)</td>
</tr>
<tr>
<td>size</td>
<td>Collision cylinder dimensions vector (x,y,z).</td>
</tr>
</tbody>
</table>

### Area Table Keys

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>box_max</td>
<td>Max vector of bounding box, must be specified if type is AREA_BOX</td>
</tr>
<tr>
<td>box_min</td>
<td>Min vector of bounding box, must be specified if type is AREA_BOX</td>
</tr>
<tr>
<td>falloff</td>
<td>Ellipsoidal falloff dimensions; 0,0,0 specifies no falloff</td>
</tr>
<tr>
<td>gravity</td>
<td>Gravity vector inside the physical area</td>
</tr>
<tr>
<td>height</td>
<td>Height of the 2D area (AREA_SHAPE), relative to the minimal Z in the points table</td>
</tr>
<tr>
<td>points</td>
<td>A table that contains an indexed collection of vectors in local entity space that define the 2D shape of the area (AREA_SHAPE)</td>
</tr>
<tr>
<td>radius</td>
<td>Radius of the area sphere; must be specified if type is AREA_SPHERE.</td>
</tr>
<tr>
<td>type</td>
<td>Type of the area, valid values are: AREA_SPHERE, AREA_BOX, AREA_GEOMETRY, or AREA_SHAPE</td>
</tr>
<tr>
<td>uniform</td>
<td>Same direction in every point, or always point to the center.</td>
</tr>
</tbody>
</table>

### PhysicalizeAttachment

**Syntax**

```
Entity.PhysicalizeAttachment(int characterSlot, const char* attachmentName, bool physicalize)
```

### PhysicalizeSlot

**Syntax**

```
Entity.PhysicalizeSlot(int slot, SmartScriptTable physicsParams)
```

### PlayFacialAnimation

**Syntax**

```
Entity.PlayFacialAnimation(char* name, bool looping)
```
PreLoadParticleEffect

Pre-loads a particle effect.

Syntax

```
Entity.PreLoadParticleEffect(const char *sEffectName)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sEffectName</td>
<td>Name of the particle effect (Ex: &quot;explosions/rocket&quot;).</td>
</tr>
</tbody>
</table>

ProcessBroadcastEvent

Syntax

```
Entity.ProcessBroadcastEvent()
```

RagDollize

Syntax

```
Entity.RagDollize(int slot)
```

ReattachSoftEntityVtx

Syntax

```
Entity.ReattachSoftEntityVtx(ScriptHandle entityId, int partId)
```

RedirectAnimationToLayer0

Syntax

```
Entity.RedirectAnimationToLayer0(int characterSlot, bool redirect)
```

RegisterForAreaEvents

Registers the script proxy so that it receives area events for this entity.

Syntax

```
Entity.RegisterForAreaEvents(int enable)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>Specify 0 to disable, or any other value to enable.</td>
</tr>
</tbody>
</table>
**RemoveAllLinks**
Removes all links of an entity.

**Syntax**

```
Entity.RemoveAllLinks()
```

**Returns:** nothing

**RemoveAuxAudioProxy**
Removes the AuxAudioProxy corresponding to the passed ID from the parent EntityAudioProxy.

**Syntax**

```
Entity.RemoveAuxAudioProxy(ScriptHandle const hAudioProxyLocalID)
```

**Returns:** nil

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAudioProxyLocalID</td>
<td>The ID of the AuxAudioProxy to be removed from the parent EntityAudioProxy.</td>
</tr>
</tbody>
</table>

**RemoveDecals**

**Syntax**

```
Entity.RemoveDecals()
```

**RemoveLink**
Removes an outgoing link from the entity.

**Syntax**

```
Entity.RemoveLink(const char *name)
```

**Returns:** nothing

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the link to remove.</td>
</tr>
</tbody>
</table>

(�ptional) ith
If specified, the \(\text{i}\)th link with the name specified that will be removed. Default value: 0 (first link with given name)

**RenderAlways**
Enables 'always render' on the render node, skipping any kind of culling.
Syntax

```cpp
Entity.RenderAlways(int enable)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>Specify 0 to disable, or any other value to enable.</td>
</tr>
</tbody>
</table>

**RenderShadow**

Syntax

```cpp
Entity.RenderShadow()
```

**ReplaceMaterial**

Syntax

```cpp
Entity.ReplaceMaterial(int slot, const char *name, const char *replacement)
```

**ResetAnimation**

Syntax

```cpp
Entity.ResetAnimation(int characterSlot, int layer)
```

**ResetAttachment**

Syntax

```cpp
Entity.ResetAttachment(int characterSlot, const char *attachmentName)
```

**ResetMaterial**

Syntax

```cpp
Entity.ResetMaterial(int slot)
```

**ResetPhysics**

Syntax

```cpp
Entity.ResetPhysics()
```

**SelectPipe**

Syntax

```cpp
Entity.SelectPipe()
```
SetAIName

Syntax

Entity.SetAIName()

SetAngles

Sets the angle of the entity.

Syntax

Entity.SetAngles(Ang3 vAngles)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vAngles</td>
<td>Angle vector.</td>
</tr>
</tbody>
</table>

SetAnimateOffScreenShadow

Syntax

Entity.SetAnimateOffScreenShadow(bool bAnimateOffScreenShadow)

SetAnimationBlendOut

Syntax

Entity.SetAnimationBlendOut(int characterSlot, int layer, float blendOut)

SetAnimationEvent

Syntax

Entity.SetAnimationEvent(int nSlot, const char *sAnimation)

SetAnimationFlip

Syntax

Entity.SetAnimationFlip(int characterSlot, Vec3 flip)

SetAnimationKeyEvent

Syntax

Entity.SetAnimationKeyEvent(nSlot, sAnimation, nFrameID, sEvent)
**SetAnimationSpeed**

**Syntax**

```
Entity.SetAnimationSpeed(int characterSlot, int layer, float speed)
```

**SetAnimationTime**

**Syntax**

```
Entity.SetAnimationTime(int nSlot, int nLayer, float fTime)
```

**SetAttachmentAngles**

**Syntax**

```
Entity.SetAttachmentAngles(int characterSlot, const char *attachmentName, Vec3 angles)
```

**SetAttachmentCGF**

**Syntax**

```
Entity.SetAttachmentCGF(int characterSlot, const char *attachmentName, const char* filePath)
```

**SetAttachmentDir**

**Syntax**

```
Entity.SetAttachmentDir(int characterSlot, const char *attachmentName, Vec3 dir, bool worldSpace)
```

**SetAttachmentEffect**

**Syntax**

```
Entity.SetAttachmentEffect(int characterSlot, const char *attachmentName, const char *effectName, Vec3 offset, Vec3 dir, float scale, int flags)
```

**SetAttachmentLight**

**Syntax**

```
Entity.SetAttachmentLight(int characterSlot, const char *attachmentName, SmartScriptTable lightTable, int flags)
```

**SetAttachmentObject**

**Syntax**

```
Entity.SetAttachmentObject(int characterSlot, const char *attachmentName, ScriptHandle entityId, int slot, int flags)
```
**SetAttachmentPos**

**Syntax**

```
Entity.SetAttachmentPos(int characterSlot, const char *attachmentName, Vec3 pos)
```

**SetAudioEnvironmentID**

Sets the ID of the audio environment that an entity will specify for the entities that it contains.

**Syntax**

```
Entity.SetAudioEnvironmentID(ScriptHandle const hAudioEnvironmentID)
```

**Returns:** nil

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAudioEnvironmentID</td>
<td>audio environment ID</td>
</tr>
</tbody>
</table>

**SetAudioObstructionCalcType**

Set the Audio Obstruction/Occlusion calculation type on the underlying GameAudioObject.

**Syntax**

```
Entity.SetAudioObstructionCalcType(int const nObstructionCalcType, ScriptHandle const hAudioProxyLocalID)
```

**Returns:** nil

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nObstructionCalcType</td>
<td>Obstruction/Occlusion calculation type; Possible values: 0 - ignore Obstruction/Occlusion 1 - use single physics ray 2 - use multiple physics rays (currently 5 per object)</td>
</tr>
<tr>
<td>hAudioProxyLocalID</td>
<td>The ID of the AuxAudioProxy that is local to the EntityAudioProxy. To address the default AuxAudioProxy, pass 1. To address all AuxAudioProxy instances, pass 0.</td>
</tr>
</tbody>
</table>

**SetAudioProxyOffset**

Set offset on the audio proxy attached to the entity.

**Syntax**

```
Entity.SetAudioProxyOffset(Vec3 const vOffset, ScriptHandle const hAudioProxyLocalID)
```
Returns: nil

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vOffset</td>
<td>The offset vector</td>
</tr>
<tr>
<td>hAudioProxyLocalID</td>
<td>The ID of the AuxAudioProxy that is local to the EntityAudioProxy. To address the default AuxAudioProxy, pass 1. To address all AuxAudioProxy instances, pass 0.</td>
</tr>
</tbody>
</table>

SetAudioRtpcValue

Set the specified audio RTPC to the specified value on the current entity.

Syntax

Entity.SetAudioRtpcValue(ScriptHandle const hRtpcID, float const fValue, ScriptHandle const hAudioProxyLocalID)

Returns: nil

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hRtpcID</td>
<td>The audio RTPC ID handle</td>
</tr>
<tr>
<td>fValue</td>
<td>The RTPC value</td>
</tr>
<tr>
<td>hAudioProxyLocalID</td>
<td>The ID of the AuxAudioProxy that is local to the EntityAudioProxy. To address the default AuxAudioProxy, pass 1. To address all AuxAudioProxy instances, pass 0.</td>
</tr>
</tbody>
</table>

SetAudioSwitchState

Set the specified audio switch to the specified state on the current Entity.

Syntax

Entity.SetAudioSwitchState(ScriptHandle const hSwitchID, ScriptHandle const hSwitchStateID, ScriptHandle const hAudioProxyLocalID)

Returns: nil

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hSwitchID</td>
<td>The audio switch ID handle</td>
</tr>
<tr>
<td>hSwitchStateID</td>
<td>The switch state ID handle</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>hAudioProxyLocalID</td>
<td>The ID of the AuxAudioProxy that is local to the EntityAudioProxy. To address the default AuxAudioProxy, pass 1. To address all AuxAudioProxy instances, pass 0.</td>
</tr>
</tbody>
</table>

**SetCharacterPhysicParams**

**Syntax**

Entity.SetCharacterPhysicParams()

**SetCloudMovementProperties**

Sets the cloud movement properties.

**Syntax**

Entity.SetCloudMovementProperties(int nSlot, SmartScriptTable table)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>Slot identifier.</td>
</tr>
<tr>
<td>table</td>
<td>Table property for the cloud movement.</td>
</tr>
</tbody>
</table>

**SetColliderMode**

**Syntax**

Entity.SetColliderMode(int mode)

**SetCurrentAudioEnvironments**

Sets the correct audio environment amounts based on the entity's position in the world.

**Syntax**

Entity.SetCurrentAudioEnvironments()

**Returns:** nil

**SetDefaultIdleAnimations**

**Syntax**

Entity.SetDefaultIdleAnimations()

**SetDirectionVector**

**Syntax**
Entity.SetDirectionVector(Vec3 dir)

**SetEnvironmentFadeDistance**

Sets the distance over which this entity fades the audio environment for all approaching entities.

**Syntax**

Entity.SetEnvironmentFadeDistance(float const fEnvironmentFadeDistance)

**Returns:** nil

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fEnvironmentFadeDistance</td>
<td>The fade distance in meters.</td>
</tr>
</tbody>
</table>

**SetFadeDistance**

Sets the distance at which this entity executes fade calculations.

**Syntax**

Entity.SetFadeDistance(float const fFadeDistance)

**Returns:** nil

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fFadeDistance</td>
<td>The fade distance in meters.</td>
</tr>
</tbody>
</table>

**SetFlags**

Mode: 0: or 1: and 2: xor

**Syntax**

Entity.SetFlags(int flags, int mode)

**SetFlagsExtended**

Mode: 0: or 1: and 2: xor

**Syntax**

Entity.SetFlagsExtended(int flags, int mode)

**SetGeomCacheDrawing**

Activates or deactivates geom cache drawing.
Syntax

Entity.SetGeomCacheDrawing(bool active)

**SetGeomCacheParams**
Sets geometry cache parameters.

Syntax

Entity.SetGeomCacheParams(bool looping, const char *standIn, const char *standInMaterial, const char *firstFrameStandIn, const char* firstFrameStandInMaterial, const char* lastFrameStandIn, const char* lastFrameStandInMaterial, float standInDistance, float streamInDistance)

**SetGeomCachePlaybackTime**
Sets the playback time.

Syntax

Entity.SetGeomCachePlaybackTime(float time)

**SetGeomCacheStreaming**
Activates/deactivates geom cache streaming.

Syntax

Entity.SetGeomCacheStreaming(bool active, float time)

**SetLightColorParams**
changes the color related params of an existing light.

Syntax

Entity.SetLightColorParams(int nSlot, Vec3 color, float specular_multiplier)

**SetLinkTarget**
Specifies the entity that an existing link shall target. Use this function to change the target of an existing link.

Syntax

Entity.SetLinkTarget(const char *name, ScriptHandle targetId)

**Returns:** nothing

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the link that shall target given entity.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>targetId</td>
<td>The ID of the entity the link shall target. Pass in NULL_ENTITY to make the link no longer target an entity.</td>
</tr>
<tr>
<td>(optional) ith</td>
<td>If specified, the (&lt;i&gt;th) link with given name that targets the specified entity. Default value: 0 (first link with given name)</td>
</tr>
</tbody>
</table>

**SetLocalAngles**

**Syntax**

```c
Entity.SetLocalAngles(Ang3 vAngles)
```

**SetLocalBBox**

**Syntax**

```c
Entity.SetLocalBBox(Vec3 vMin, Vec3 vMax)
```

**SetLocalPos**

**Syntax**

```c
Entity.SetLocalPos(Vec3 vPos)
```

**SetLocalScale**

**Syntax**

```c
Entity.SetLocalScale(float fScale)
```

**SetLodRatio**

**Syntax**

```c
Entity.SetLodRatio()
```

**SetMaterial**

**Syntax**

```c
Entity.SetMaterial()
```

**SetMaterialFloat**

Change material parameter.

**Syntax**

```c
Entity.SetMaterialFloat(int slot, int nSubMtlId, const char *sParamName, float fValue)
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slot</td>
<td>ID of the slot on which to change material.</td>
</tr>
<tr>
<td>nSubMtlId</td>
<td>Specify sub-material by ID.</td>
</tr>
<tr>
<td>sParamName</td>
<td>Name of the material parameter.</td>
</tr>
<tr>
<td>fValue</td>
<td>New material parameter value.</td>
</tr>
</tbody>
</table>

### SetMaterialVec3

**Syntax**

```c
Entity.SetMaterialVec3(int slot, int nSubMtlId, const char *sParamName, Vec3 fValue)
```

### SetName

**Syntax**

```c
Entity.SetName()
```

### SetParentSlot

**Syntax**

```c
Entity.SetParentSlot(int child, int parent)
```

### SetPhysicParams

**Syntax**

```c
Entity.SetPhysicParams()
```

### SetPos

Sets the position of the entity.

**Syntax**

```c
Entity.SetPos(Vec3 vPos)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vPos</td>
<td>Position vector.</td>
</tr>
</tbody>
</table>

### SetPublicParam

Sets a shader parameter.
** entity.SetPublicParam() **

** Parameters **

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>paramName</td>
<td>The name of the shader parameter.</td>
</tr>
<tr>
<td>value</td>
<td>The new value of the parameter.</td>
</tr>
</tbody>
</table>

** SetRegisterInSectors **

** entity.SetRegisterInSectors() **

** SetScale **

Sets the scaling value for the entity.

** Syntax **

** entity.SetScale(float fScale) **

** Parameters **

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fScale</td>
<td>The scale amount.</td>
</tr>
</tbody>
</table>

** SetScriptUpdateRate **

** entity.SetScriptUpdateRate(int nMillis) **

** SetSelfAsLightCasterException **

Makes the entity render node a caster exception for the light loaded in nLightSlot.

** Syntax **

** entity.SetSelfAsLightCasterException(int nLightSlot) **

** Parameters **

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nLightSlot</td>
<td>Slot where the light is loaded.</td>
</tr>
</tbody>
</table>
**SetSlotAngles**
Sets the slot angles.

**Syntax**

```c
Entity.SetSlotAngles(int nSlot, Ang3 v)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>nSlot identifier.</td>
</tr>
<tr>
<td>v</td>
<td>Angle to be set.</td>
</tr>
</tbody>
</table>

**SetSlotHud3D**

Setup flags for use as 3D HUD entity.

**Syntax**

```c
Entity.SetSlotHud3D(int nSlot)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>Slot identifier.</td>
</tr>
</tbody>
</table>

**SetSlotPos**

Sets the slot position.

**Syntax**

```c
Entity.SetSlotPos(int slot, Vec3 v)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slot</td>
<td>slot identifier.</td>
</tr>
<tr>
<td>v</td>
<td>Position to be set.</td>
</tr>
</tbody>
</table>

**SetSlotPosAndDir**

Sets the slot position and direction.

**Syntax**

```c
Entity.SetSlotPosAndDir(int nSlot, Vec3 pos, Vec3 dir)
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>nSlot identifier.</td>
</tr>
<tr>
<td>pos</td>
<td>Position to be set.</td>
</tr>
<tr>
<td>dir</td>
<td>Direction to be set.</td>
</tr>
</tbody>
</table>

### SetSlotScale

Sets the slot scale amount.

**Syntax**

```c
Entity.SetSlotScale(int nSlot, float fScale)
```

### SetSlotWorldTM

Sets the World TM (transformation matrix) of the slot.

**Syntax**

```c
Entity.SetSlotWorldTM(int nSlot, Vec3 pos, Vec3 dir)
```

### setStateClientside

**Syntax**

```c
Entity.SetStateClientside()
```

### SetTimer

**Syntax**

```c
```

872
SetTimer

Syntax

Entity.SetTimer()

SetTriggerBBox

Syntax

Entity.SetTriggerBBox(Vec3 vMin, Vec3 vMax)

SetUpdatePolicy

Changes the update policy for the entity. Update policy controls when an entity becomes active or inactive (for example, when visible or when in close proximity).

**Note**
Because all active entities are updated every frame, having too many active entities can affect performance.

Syntax

Entity.SetUpdatePolicy(int nUpdatePolicy)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nUpdatePolicy</td>
<td>Update policy constant. See the following table for possible values.</td>
</tr>
</tbody>
</table>

**nUpdatePolicy Possible Values**

<table>
<thead>
<tr>
<th>Update Policy</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTITY_UPDATE_NEVER</td>
<td>Never update this entity.</td>
</tr>
<tr>
<td>ENTITY_UPDATE_IN_RANGE</td>
<td>Activate entity when in specified radius.</td>
</tr>
<tr>
<td>ENTITY_UPDATE_POT_VISIBLE</td>
<td>Activate entity when potentially visible.</td>
</tr>
<tr>
<td>ENTITY_UPDATE_VISIBLE</td>
<td>Activate entity when visible in frustum.</td>
</tr>
<tr>
<td>ENTITY_UPDATE_PHYSICS</td>
<td>Activate entity when physics awakes, deactivate when physics go to sleep.</td>
</tr>
<tr>
<td>ENTITY_UPDATE_PHYSICS_VISIBLE</td>
<td>ENTITY_UPDATE_PHYSICS, but also activates when visible.</td>
</tr>
<tr>
<td>ENTITY_UPDATE_ALWAYS</td>
<td>Entity is always active and updated every frame.</td>
</tr>
</tbody>
</table>

**Note**
For update policies that require a radius, use SetUpdateRadius (p. 873).

SetUpdateRadius

Syntax

Entity.SetUpdateRadius()
SetVelocity

Syntax

Entity.SetVelocity(Vec3 velocity)

SetVelocityEx

Syntax

Entity.SetVelocityEx(Vec3 velocity, Vec3 angularVelocity)

SetViewDistanceMultiplier

Set the view distance multiplier.

Syntax

Entity.SetViewDistanceMultiplier()

SetViewDistUnlimited

Syntax

Entity.SetViewDistUnlimited()

SetVolumeObjectMovementProperties

Sets the properties of the volume object movement.

Syntax

Entity.SetVolumeObjectMovementProperties(int nSlot, SmartScriptTable table)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>Slot identifier.</td>
</tr>
<tr>
<td>table</td>
<td>Table with volume object properties.</td>
</tr>
</tbody>
</table>

SetWorldAngles

Syntax

Entity.SetWorldAngles(Ang3 vAngles)

SetWorldPos

Syntax
Entity.SetWorldPos(Vec3 vPos)

**SetWorldScale**

**Syntax**

Entity.SetWorldScale(float fScale)

**StartAnimation**

**Syntax**

Entity.StartAnimation()

**StopAnimation**

**Syntax**

Entity.StopAnimation(int characterSlot, int layer)

**StopAudioTrigger**

Stop the audio event generated by the trigger with the specified ID on this entity.

**Syntax**

Entity.StopAudioTrigger(ScriptHandle const hTriggerID, ScriptHandle const hAudioProxyLocalID)

**Returns:** nil

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hTriggerID</td>
<td>The audio trigger ID handle</td>
</tr>
<tr>
<td>hAudioProxyLocalID</td>
<td>The ID of the AuxAudioProxy that is local to the EntityAudioProxy. To address the default AuxAudioProxy, pass 1. To address all AuxAudioProxy instances, pass 0.</td>
</tr>
</tbody>
</table>

**ToGlobal**

**Syntax**

Entity.ToGlobal(int slotId, Vec3 point)

**ToLocal**

**Syntax**


Lumberyard Legacy Reference
ScriptBind Engine Functions

Entity.ToLocal(int slotId, Vec3 point)

**TriggerEvent**

**Syntax**

Entity.TriggerEvent()

**UnSeenFrames**

**Syntax**

Entity.UnSeenFrames()

**UpdateAreas**

**Syntax**

Entity.UpdateAreas()

**UpdateLightClipBounds**

Update the clip bounds of the light from the linked entities.

**Syntax**

Entity.UpdateLightClipBounds(int nSlot)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSlot</td>
<td>Slot identifier.</td>
</tr>
</tbody>
</table>

**UpdateSlotPhysics**

**Syntax**

Entity.UpdateSlotPhysics(int slot)

**VectorToGlobal**

**Syntax**

Entity.VectorToGlobal(int slotId, Vec3 dir)

**VectorToLocal**

**Syntax**

Entity.VectorToLocal(int slotId, Vec3 dir)
ScriptBind_Movie

CryLua is deprecated and will be removed in a future version of Lumberyard.

Lists C++ movie functions that can be called from Lua scripts.

AbortSequence

Aborts the specified sequence.

Syntax

```cpp
Movie.AbortSequence(const char *sSequenceName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sSequenceName</td>
<td>Sequence name.</td>
</tr>
</tbody>
</table>

PauseSequences

Pauses all the sequences.

Syntax

```cpp
Movie.PauseSequences()
```

PlaySequence

Plays the specified sequence.

Syntax

```cpp
Movie.PlaySequence(const char *sSequenceName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sSequenceName</td>
<td>Sequence name.</td>
</tr>
</tbody>
</table>

ResumeSequences

Resume all the sequences.

Syntax

```cpp
Movie.ResumeSequences()
```

StopAllCutScenes

Stops all the cut scenes.
Syntax

```
Movie.StopAllCutScenes()
```

**StopAllSequences**

Stops all the video sequences.

Syntax

```
Movie.StopAllSequences()
```

**StopSequence**

Stops the specified sequence.

Syntax

```
Movie.StopSequence(const char *sSequenceName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sSequenceName</td>
<td>Sequence name.</td>
</tr>
</tbody>
</table>

**ScriptBind_Particle**

CryLua is deprecated and will be removed in a future version of Lumberyard.

Lists C++ particle functions that you can call from Lua script.

**Attach**

Attaches an effect.

Syntax

```
Particle.Attach()
```

**CreateDecal**

Creates a decal with the specified parameters.

Syntax

```
Particle.CreateDecal(Vec3 pos, Vec3 normal, float size, float lifeTime, const char *textureName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>The decal position vector.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>normal</td>
<td>The decal normal vector.</td>
</tr>
<tr>
<td>size</td>
<td>The decal size, expressed as a float.</td>
</tr>
<tr>
<td>lifeTime</td>
<td>The decal lifetime, expressed as a float.</td>
</tr>
<tr>
<td>textureName</td>
<td>The name of the texture.</td>
</tr>
</tbody>
</table>

**CreateEffect**

Creates a particle effect.

**Syntax**

```c
Particle.CreateEffect(const char *name, SmartScriptTable params)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the particle effect.</td>
</tr>
<tr>
<td>params</td>
<td>A SmartScriptTable of effect parameters.</td>
</tr>
</tbody>
</table>

**CreateMatDecal**

Creates a material decal.

**Syntax**

```c
Particle.CreateMatDecal(Vec3 pos, Vec3 normal, float size, float lifeTime, const char *materialName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>The decal position vector.</td>
</tr>
<tr>
<td>normal</td>
<td>The decal normal vector.</td>
</tr>
<tr>
<td>size</td>
<td>The decal size, expressed as a float.</td>
</tr>
<tr>
<td>lifeTime</td>
<td>The decal lifetime, expressed as a float.</td>
</tr>
<tr>
<td>materialName</td>
<td>The name of the material.</td>
</tr>
</tbody>
</table>

**DeleteEffect**

Deletes the specified particle effect.

**Syntax**

```c
Particle.DeleteEffect(const char *name)
```
### Detach

Detaches an effect.

**Syntax**

```cpp
Particle.Detach()
```

### IsEffectAvailable

Checks if the specified particle effect is available.

**Syntax**

```cpp
Particle.IsEffectAvailable(const char *name)
```

- **Parameter**
  - `name` - The name of the particle effect to check for availability.

### SpawnEffect

Spawns an effect.

**Syntax**

```cpp
Particle.SpawnEffect(const char *effectName, Vec3 pos, Vec3 dir)
```

- **Parameter**
  - `effectName` - The name of the effect to spawn.
  - `pos` - The position vector of the effect.
  - `dir` - The direction vector of the effect.

### SpawnEffectLine

Spawns an effect line.

**Syntax**

```cpp
Particle.SpawnEffectLine(const char *effectName, Vec3 startPos, Vec3 endPos, Vec3 dir, float scale, int slices)
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>effectName</td>
<td>The name of the effect.</td>
</tr>
<tr>
<td>startPos</td>
<td>The start position of the effect.</td>
</tr>
<tr>
<td>endPos</td>
<td>The end position of the effect.</td>
</tr>
<tr>
<td>dir</td>
<td>The direction of the effect.</td>
</tr>
<tr>
<td>scale</td>
<td>The scale value of the effect, expressed as a float.</td>
</tr>
<tr>
<td>slices</td>
<td>The number of slices.</td>
</tr>
</tbody>
</table>

**SpawnParticles**

Spawns a particle effect.

**Syntax**

```
Particle.SpawnParticles(SmartScriptTable params, Vec3 pos, Vec3 dir)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>params</td>
<td>A SmartScriptTable of particle effect parameters.</td>
</tr>
<tr>
<td>pos</td>
<td>The position vector of the particle effect.</td>
</tr>
<tr>
<td>dir</td>
<td>The direction vector of the particle effect.</td>
</tr>
</tbody>
</table>

**ScriptBind_Physics**

CryLua is deprecated and will be removed in a future version of Lumberyard.

Lists C++ physics functions that you can call from Lua script.

**RayTraceCheck**

Checks if a ray segment intersects anything from its source to its destination.

**Syntax**

```
Physics.RayTraceCheck(Vec3 src, Vec3 dst, ScriptHandle skipEntityId1, ScriptHandle skipEntityId2)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>The origin point of the ray segment.</td>
</tr>
<tr>
<td>dst</td>
<td>The end point of the ray segment.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------------------</td>
</tr>
<tr>
<td>skipEntityId1</td>
<td>Entity ID to skip when checking for intersection.</td>
</tr>
<tr>
<td>skipEntityId2</td>
<td>Entity ID to skip when checking for intersection.</td>
</tr>
</tbody>
</table>

**RayWorldIntersection**

Checks if a ray segment intersects anything from its source to its destination.

**Syntax**

```c
Physics.RaycastWorldIntersection(Vec3 vPos, Vec3 vDir, int nMaxHits, int iEntTypes [, skipEntityId1 [, skipEntityId2]])
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vPos</td>
<td>The origin point of the ray.</td>
</tr>
<tr>
<td>vDir</td>
<td>The direction of the ray.</td>
</tr>
<tr>
<td>nMaxHits</td>
<td>The maximum number of hits to return, sorted in nearest to farthest order.</td>
</tr>
<tr>
<td>iEntTypes</td>
<td>A bitmask of physical entity types. The ray intersects only with entities that the mask specifies (ent_all,...).</td>
</tr>
<tr>
<td>skipEntityId1</td>
<td>Optional. An entity ID to skip when checking for intersection.</td>
</tr>
<tr>
<td>skipEntityId2</td>
<td>Optional. An entity ID to skip when checking for intersection.</td>
</tr>
</tbody>
</table>

**RegisterExplosionCrack**

Registers a new crack for a breakable object.

**Syntax**

```c
Physics.RegisterExplosionCrack(const char *sGeometryFile, int nIdMaterial)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sGeometryFile</td>
<td>The name of the static geometry file for the crack (CGF).</td>
</tr>
<tr>
<td>nMaterialId</td>
<td>The ID of the breakable material to which the crack is applied.</td>
</tr>
</tbody>
</table>

**RegisterExplosionShape**

Registers a new explosion shape from static geometry.
**Note**

RegisterExplosionShape applies only physical forces; it does not apply any game related explosion damages.

**Syntax**

```cpp
Physics.RegisterExplosionShape(RegisterExplosionShape(IFunctionHandler *pH, const char *sGeometryFile, float fSize, int nIdMaterial, float fProbability, const char *sSplintersFile, float fSplintersOffset, const char *sSplintersCloudEffect)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sGeometryFile</td>
<td>The name of the static geometry file (CGF).</td>
</tr>
<tr>
<td>fSize</td>
<td>The scale for the static geometry.</td>
</tr>
<tr>
<td>nIdMaterial</td>
<td>The ID of the breakable material on which the shape is applied.</td>
</tr>
<tr>
<td>fProbability</td>
<td>The preference ratio for using this shape instead of other registered shapes.</td>
</tr>
<tr>
<td>sSplintersFile</td>
<td>The name of a CGF file that contains additional non-physicalized splinters to place on cut surfaces.</td>
</tr>
<tr>
<td>fSplintersOffset</td>
<td>The lower splinters position in relation to the upper one.</td>
</tr>
<tr>
<td>sSplintersCloudEffect</td>
<td>The particle effect when the splinters constraint breaks.</td>
</tr>
</tbody>
</table>

**SamplePhysEnvironment**

Find the physical entities touched by a sphere.

**Syntax**

```cpp
Physics.SamplePhysEnvironment(pt, r [, objtypes])
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pt</td>
<td>The center of the sphere.</td>
</tr>
<tr>
<td>r</td>
<td>The radius of the sphere.</td>
</tr>
<tr>
<td>objtypes</td>
<td>Optional. The types of physical entities that the sphere touches.</td>
</tr>
</tbody>
</table>

**SimulateExplosion**

Simulates a physical explosion.

**Note**

SimulateExplosion applies only physical forces; it does not apply any game related explosion damages.
Syntax

```
Physics.SimulateExplosion(SmartScriptTable explosionParams)
```

`explosionParams` is a SmartScriptTable whose elements are as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>The epicenter of the explosion.</td>
</tr>
<tr>
<td>radius</td>
<td>The radius of the explosion.</td>
</tr>
<tr>
<td>direction</td>
<td>The direction of the explosion impulse.</td>
</tr>
<tr>
<td>impulse_pos</td>
<td>The position of the explosion impulse. This value can be different from the epicenter of the explosion.</td>
</tr>
<tr>
<td>impulse_pressure</td>
<td>The pressure of the explosion impulse at the specified radius from the epicenter.</td>
</tr>
<tr>
<td>rmin</td>
<td>The minimal radius of the explosion. At this radius, full pressure is applied.</td>
</tr>
<tr>
<td>rmax</td>
<td>The maximum radius of the explosion. At this radius, the impulse pressure is close to zero.</td>
</tr>
<tr>
<td>hole_size</td>
<td>The size of the hole that the explosion creates in breakable objects.</td>
</tr>
</tbody>
</table>

**ScriptBind_Script**

CryLua is deprecated and will be removed in a future version of Lumberyard.

Lists C++ script-related functions that you can call from Lua script.

**DumpLoadedScripts**

Dumps all loaded scripts.

**Syntax**

```
Script.DumpLoadedScripts()
```

**KillTimer**

Stops a timer set by the `Script.SetTimer` function.

**Syntax**

```
Script.KillTimer(ScriptHandle nTimerId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nTimerId</td>
<td>The ID of the timer returned by the <code>Script.SetTimer</code> function.</td>
</tr>
</tbody>
</table>
**LoadScript**

Loads the specified script.

**Syntax**

Script.LoadScript(scriptName)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scriptName</td>
<td>The name of the script to load.</td>
</tr>
</tbody>
</table>

**ReloadEntityScript**

Reloads the specified entity script.

**Syntax**

Script.ReloadEntityScript(const char *className)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>className</td>
<td>Name of the entity script.</td>
</tr>
</tbody>
</table>

**ReloadScript**

Reload the script.

**Syntax**

Script.ReloadScript(scriptName)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scriptName</td>
<td>The name of the script to reload.</td>
</tr>
</tbody>
</table>

**ReloadScripts**

Reloads all the scripts.

**Syntax**

Script.ReloadScripts()

**SetTimer**

Sets a script timer. When the timer expires, SetTimer calls the Lua function specified.

**Syntax**

Script.SetTimer(int nMilliseconds, HSCRIPTFUNCTION hFunc)
Returns the ID assigned to the timer or nil if no ID was specified.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nMilliseconds</td>
<td>Delay of the trigger in milliseconds.</td>
</tr>
<tr>
<td>luaFunction</td>
<td>The Lua function to call. If userData is specified, luaFunction must be in the format:</td>
</tr>
<tr>
<td></td>
<td>LuaCallback = function(userData, nTimerId)</td>
</tr>
<tr>
<td></td>
<td>-- function body</td>
</tr>
<tr>
<td></td>
<td>end;</td>
</tr>
<tr>
<td>userData</td>
<td>Optional. Specifies a user defined table. If userData is specified, the table is passed as the first argument of the callback function.</td>
</tr>
<tr>
<td>bUpdateDuringPause</td>
<td>Optional. The timer is updated and triggered even if the game is in pause mode.</td>
</tr>
</tbody>
</table>

**SetTimerForFunction**

Sets a timer for the specified function.

**Syntax**

```lua
Script.SetTimerForFunction(int nMilliseconds, const char *sFunctionName)
```

Returns the ID assigned to the timer, or nil if no ID was specified.

This function has the same parameters as the `SetTimer` function.

**UnloadScript**

Unloads the specified script.

**Syntax**

```lua
Script.UnloadScript(scriptName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scriptName</td>
<td>The name of the script to unload.</td>
</tr>
</tbody>
</table>

**ScriptBind_Sound**

CryLua is deprecated and will be removed in a future version of Lumberyard.
Lists C++ sound functions that can be called from Lua scripts.

**GetAudioEnvironmentID**

Get the audio environment TAudioEnvironmentID (wrapped into a ScriptHandle).

**Syntax**

\[
\text{Sound.GetAudioEnvironmentID(const char* const sEnvironmentName)}
\]

**Returns:** ScriptHandle with the TAudioEnvironmentID value, or nil if the sEnvironmentName is not found.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sEnvironmentName</td>
<td>The unique name of an audio environment.</td>
</tr>
</tbody>
</table>

**GetAudioRtpcID**

Get the RTPC TAudioControlID (wrapped into a ScriptHandle).

**Syntax**

\[
\text{Sound.GetAudioRtpcID(const char* const sRtpcName)}
\]

**Returns:** ScriptHandle with the TAudioControlID value, or nil if the sRtpcName is not found.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sRtpcName</td>
<td>The unique name of an audio RTPC.</td>
</tr>
</tbody>
</table>

**GetAudioSwitchID**

Get the switch TAudioControlID (wrapped into a ScriptHandle).

**Syntax**

\[
\text{Sound.GetAudioSwitchID(const char* const sSwitchName)}
\]

**Returns:** ScriptHandle with the TAudioControlID value, or nil if the sSwitchName is not found.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sSwitchName</td>
<td>The unique name of an audio switch.</td>
</tr>
</tbody>
</table>

**GetAudioSwitchStateID**

Get the SwitchState TAudioSwitchStateID (wrapped into a ScriptHandle).

**Syntax**

\[
\text{Sound.GetAudioSwitchStateID(const char* const sSwitchName)}
\]
Sound.GetAudioSwitchStateID(const ScriptHandle hSwitchID, const char* const sSwitchStateName)

Returns: ScriptHandle with the TAUDIO_SWITCH_STATE_ID value, or nil if the sSwitchStateName is not found.

Parameter | Description
--- | ---
sSwitchStateName | The unique name of an audio switch state.

GetAudioTriggerID

Get the trigger TAudioControlID (wrapped into a ScriptHandle).

Syntax

Sound.GetAudioTriggerID(const char* const sTriggerName)

Returns: ScriptHandle with the TAUDIO_CONTROL_ID value, or nil if the sTriggerName is not found.

Parameter | Description
--- | ---
sTriggerName | The unique name of an audio trigger.

SetAudioRtpcValue

Globally sets the specified audio RTPC to the specified value.

Syntax

Sound.SetAudioRtpcValue( hRtpcID, fValue )

Returns: nil

Parameter | Description
--- | ---
hRtpcID | The audio RTPC ID handle.
fValue | The RTPC value.

ScriptBind_System

CryLua is deprecated and will be removed in a future version of Lumberyard.

This class implements Lua script functions that expose system functionalities.

ActivatePortal

Activates or deactivates a portal.
Syntax

System.ActivatePortal(Vec3 vPos, bool bActivate, ScriptHandle nID)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vPos</td>
<td>Position vector.</td>
</tr>
<tr>
<td>bActivate</td>
<td>True to activate the portal, false to deactivate.</td>
</tr>
<tr>
<td>nID</td>
<td>Entity identifier.</td>
</tr>
</tbody>
</table>

AddCCommand

Adds a C command to the system.

Syntax

System.AddCCommand(const char* sCCommandName, const char* sCommand, const char* sHelp)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sCCommandName</td>
<td>C command name.</td>
</tr>
<tr>
<td>sCommand</td>
<td>Command string.</td>
</tr>
<tr>
<td>sHelp</td>
<td>Help for the command usage.</td>
</tr>
</tbody>
</table>

ApplicationTest

Test the application with the specified parameters.

Syntax

System.ApplicationTest(const char* pszParam)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pszParam</td>
<td>Parameters.</td>
</tr>
</tbody>
</table>

Break

Breaks the application with a fatal error message.

Syntax

System.Break()
**BrowseURL**

Browses a URL address.

**Syntax**

```
System.BrowseURL(const char* szURL)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>szURL</td>
<td>URL string.</td>
</tr>
</tbody>
</table>

**CheckHeapValid**

Checks the heap validity.

**Syntax**

```
System.CheckHeapValid(const char* name)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name string. The default is &lt;noname&gt;.</td>
</tr>
</tbody>
</table>

**ClearConsole**

Clears the console.

**Syntax**

```
System.ClearConsole()
```

**ClearKeyState**

Clear the key state.

**Syntax**

```
System.ClearKeyState()
```

**CreateDownload**

**Syntax**

```
System.CreateDownload()
```

**DebugStats**

**Syntax**

```
System.DebugStats()
```
System.DebugStats(bool cp)

**DeformTerrain**

Deforms the terrain.

**Syntax**

```c
System.DeformTerrain()
```

**DeformTerrainUsingMat**

Deforms the terrain using material.

**Syntax**

```c
System.DeformTerrainUsingMat()
```

**Draw2DLine**

Draws a 2D line.

**Syntax**

```c
System.Draw2DLine(p1x, p1y, p2x, p2y, float r, float g, float b, float alpha)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1x</td>
<td>X value of the start point of the line.</td>
</tr>
<tr>
<td>p1y</td>
<td>Y value of the start point of the line.</td>
</tr>
<tr>
<td>p2x</td>
<td>X value of the end point of the line.</td>
</tr>
<tr>
<td>p2y</td>
<td>Y value of the end point of the line.</td>
</tr>
<tr>
<td>r</td>
<td>Red component for the label color. Default is 1.</td>
</tr>
<tr>
<td>g</td>
<td>Green component for the label color. Default is 1.</td>
</tr>
<tr>
<td>b</td>
<td>Blue component for the label color. Default is 1.</td>
</tr>
<tr>
<td>alpha</td>
<td>Alpha component for the label color. Default is 1.</td>
</tr>
</tbody>
</table>

**DrawLabel**

Draws a label with the specified parameter.

**Syntax**

```c
System.DrawLabel(Vec3 vPos, float fSize, const char* text [, float r [, float g [, float b [, float alpha]]]])
```
**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vPos</td>
<td>Position vector.</td>
</tr>
<tr>
<td>fSize</td>
<td>Size for the label.</td>
</tr>
<tr>
<td>text</td>
<td>Text of the label.</td>
</tr>
<tr>
<td>r</td>
<td>Red component for the label colour. Default is 1.</td>
</tr>
<tr>
<td>g</td>
<td>Green component for the label colour. Default is 1.</td>
</tr>
<tr>
<td>b</td>
<td>Blue component for the label colour. Default is 1.</td>
</tr>
<tr>
<td>alpha</td>
<td>Alpha component for the label colour. Default is 1.</td>
</tr>
</tbody>
</table>

**DrawLine**

Draws a line.

**Syntax**

```plaintext
System.DrawLine(Vec3 p1, Vec3 p2, float r, float g, float b, float alpha)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>Start position of the line.</td>
</tr>
<tr>
<td>p2</td>
<td>End position of the line.</td>
</tr>
<tr>
<td>r</td>
<td>Red component for the label color. Default is 1.</td>
</tr>
<tr>
<td>g</td>
<td>Green component for the label color. Default is 1.</td>
</tr>
<tr>
<td>b</td>
<td>Blue component for the label color. Default is 1.</td>
</tr>
<tr>
<td>alpha</td>
<td>Alpha component for the label color. Default is 1.</td>
</tr>
</tbody>
</table>

**DrawText**

Draws text.

**Syntax**

```plaintext
System.DrawText(float x, float y, const char* text, const char* fontName, float size, float r, float g, float b, float alpha)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>X position for the text. The default is 0.</td>
</tr>
<tr>
<td>y</td>
<td>Y position for the text. The default is 0.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>text</td>
<td>Text to be displayed. The default is an empty string.</td>
</tr>
<tr>
<td>fontName</td>
<td>Font name. The default is default.</td>
</tr>
<tr>
<td>size</td>
<td>Text size. The default is 16.</td>
</tr>
<tr>
<td>r</td>
<td>Red component for the label color. The default is 1.</td>
</tr>
<tr>
<td>g</td>
<td>Green component for the label color. The default is 1.</td>
</tr>
<tr>
<td>b</td>
<td>Blue component for the label color. The default is 1.</td>
</tr>
<tr>
<td>alpha</td>
<td>Alpha component for the label color. The default is 1.</td>
</tr>
</tbody>
</table>

**DumpMemoryCoverage**

Dumps memory coverage.

**Syntax**

```
System.DumpMemoryCoverage()
```

This function is useful for investigating memory fragmentation. When `System.DumpMemoryCoverage()` is called from the console, `DumpMemoryCoverage` adds a line to the `MemoryCoverage.bmp` file, which is generated the first time there is a maximum line count.

**DumpMemStats**

Dumps memory statistics.

**Syntax**

```
System.DumpMemStats(bUseKB)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bUseKB</td>
<td>True to use KB, false otherwise. The default is false.</td>
</tr>
</tbody>
</table>

**DumpMMStats**

Dumps the MM statistics.

**Syntax**

```
System.DumpMMStats()
```

**DumpWinHeaps**

Dumps windows heaps.
Syntax

```
System.DumpWinHeaps()
```

**EnableOceanRendering**

Enables/disables ocean rendering.

**Syntax**

```
System.EnableOceanRendering()
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bOcean</td>
<td>True to activate the ocean rendering, false to deactivate it.</td>
</tr>
</tbody>
</table>

**EnumAAFormats**

Enumerates multisample anti-aliasing formats.

**Syntax**

```
System.EnumAAFormats()
```

**EnumDisplayFormats**

Enumerates display formats.

**Syntax**

```
System.EnumDisplayFormats()
```

**Error**

Shows a message text with the error severity.

**Syntax**

```
System.Error(const char* sParam)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sParam</td>
<td>Text to be logged. The default is an empty string.</td>
</tr>
</tbody>
</table>

**ExecuteCommand**

Executes a command.

**Syntax**
System.ExecuteCommand(const char* szCmd)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>szCmd</td>
<td>Command string.</td>
</tr>
</tbody>
</table>

GetConfigSpec

Gets the config specification.

Syntax

System.GetConfigSpec()

GetCurrAsyncTime

Gets the current asynchronous time.

Syntax

System.GetCurrAsyncTime()

GetCurrTime

Gets the current time.

Syntax

System.GetCurrTime()

GetCVar

Gets the value of a console variable.

Syntax

System.GetCVar(const char* sCVarName)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sCVarName</td>
<td>Name of the variable.</td>
</tr>
</tbody>
</table>

GetEntities

Returns a table with all the entities currently present in a level.

Syntax

System.GetEntities(Vec3 center, float radius)
**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>center</td>
<td>Center position vector for the area where to get entities. The default is (0, 0, 0).</td>
</tr>
<tr>
<td>radius</td>
<td>Radius of the area. The default is 0.</td>
</tr>
</tbody>
</table>

**GetEntitiesByClass**

Gets all the entities of the specified class.

**Syntax**

```cpp
System.GetEntitiesByClass(const char* EntityClass)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntityClass</td>
<td>Entity class name.</td>
</tr>
</tbody>
</table>

**GetEntitiesInSphere**

Gets all the entities contained into the specified sphere.

**Syntax**

```cpp
System.GetEntitiesInSphere(Vec3 center, float radius)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>center</td>
<td>center position vector for the sphere where to look at entities.</td>
</tr>
<tr>
<td>radius</td>
<td>Radius of the sphere.</td>
</tr>
</tbody>
</table>

**GetEntitiesInSphereByClass**

Gets all the entities contained into the specified sphere for the specific class name.

**Syntax**

```cpp
System.GetEntitiesInSphereByClass(Vec3 center, float radius, const char* EntityClass)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>center</td>
<td>center position vector for the sphere where to look at entities.</td>
</tr>
<tr>
<td>radius</td>
<td>Radius of the sphere.</td>
</tr>
</tbody>
</table>
### GetEntity

Gets an entity from its ID.

**Syntax**

```c
System.GetEntity(entityId)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>Entity identifier (svtNumber or ScriptHandle).</td>
</tr>
</tbody>
</table>

### GetEntityByName

Retrieve entity table for the first entity with specified name. If multiple entities with same name exist, first one found is returned.

**Syntax**

```c
System.GetEntityByName(const char *sEntityName)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sEntityName</td>
<td>Name of the entity to search.</td>
</tr>
</tbody>
</table>

### GetEntityClass

Gets an entity class from its ID.

**Syntax**

```c
System.GetEntityClass(entityId)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>Entity identifier (svtNumber or ScriptHandle).</td>
</tr>
</tbody>
</table>

### GetEntityIdByName

Retrieve entity Id for the first entity with specified name. If multiple entities with same name exist, first one found is returned.

**Syntax**

```c
System.GetEntityIdByName(const char *sEntityName)
```
System.GetEntityIdByName(const char *sEntityName)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sEntityName</td>
<td>Name of the entity to search.</td>
</tr>
</tbody>
</table>

**GetFrameID**

Gets the frame identifier.

**Syntax**

System.GetFrameID()

**GetFrameTime**

Gets the frame time.

**Syntax**

System.GetFrameTime()

**GetHDRDynamicMultiplier**

Gets the HDR dynamic multiplier.

**Syntax**

System.GetHDRDynamicMultiplier()

**GetLocalOSTime**

Gets the local operating system time.

**Syntax**

System.GetLocalOSTime()

**GetNearestEntityByClass**

Gets the nearest entity with the specified class.

**Syntax**

System.GetNearestEntityByClass(Vec3 center, float radius, const char *className)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>center</td>
<td>Center position vector for the area where to look at entities.</td>
</tr>
</tbody>
</table>
### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>radius</td>
<td>Radius of the sphere.</td>
</tr>
<tr>
<td>className</td>
<td>Entity class name.</td>
</tr>
</tbody>
</table>

### GetOutdoorAmbientColor

Gets the outdoor ambient color.

**Syntax**

```c
System.GetOutdoorAmbientColor()
```

### GetPhysicalEntitiesInBox

Gets all the entities contained into the specified area.

**Syntax**

```c
System.GetPhysicalEntitiesInBox(Vec3 center, float radius)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>center</td>
<td>Center position vector for the area where to look at entities.</td>
</tr>
<tr>
<td>radius</td>
<td>Radius of the sphere.</td>
</tr>
</tbody>
</table>

### GetPhysicalEntitiesInBoxByClass

Gets all the entities contained into the specified area for the specific class name.

**Syntax**

```c
System.GetPhysicalEntitiesInBoxByClass(Vec3 center, float radius, const char *className)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>center</td>
<td>Center position vector for the area where to look at entities.</td>
</tr>
<tr>
<td>radius</td>
<td>Radius of the sphere.</td>
</tr>
<tr>
<td>className</td>
<td>Entity class name.</td>
</tr>
</tbody>
</table>

### GetPostProcessFxParam

Gets a post processing effect parameter value.
**GetPostProcessFxParam**

Syntax

```cpp
System.GetPostProcessFxParam(const char* pszEffectParam, value)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pszEffectParam</td>
<td>Parameter for the post processing effect.</td>
</tr>
<tr>
<td>value</td>
<td>Value for the parameter (float or string).</td>
</tr>
</tbody>
</table>

**GetScreenFx**

Gets a post processing effect parameter value.

**Note**

This is a convenience wrapper function for `GetPostProcessFxParam`.

Syntax

```cpp
System.GetScreenFx(const char* pszEffectParam, value)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pszEffectParam</td>
<td>Parameter for the post processing effect.</td>
</tr>
<tr>
<td>value</td>
<td>Value for the parameter (float or string).</td>
</tr>
</tbody>
</table>

**GetSkyColor**

Retrieve color of the sky (outdoor ambient color).

Syntax

```cpp
Vec3 System.GetSkyColor()
```

**Returns:** Sky color as an \( \{x, y, z\} \) vector (x=r, y=g, z=b).

**GetSkyHighlight**

Retrieves sky highlighting parameters. See `SetSkyHighlight (p. 911)` for a description of the parameters.

Syntax

```cpp
System.GetSkyHighlight(SmartScriptTable params)
```

**GetSunColor**

Retrieve color of the sun outdoors.

Syntax
Vec3 System.GetSunColor()

**Returns:** Sun Color as an \(x, y, z\) vector \((x=r, y=g, z=b)\).

**GetSurfaceTypeIdByName**

Gets the surface type identifier by its name.

**Syntax**

```cpp
System.GetSurfaceTypeIdByName(const char* surfaceName)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>surfaceName</td>
<td>Surface name.</td>
</tr>
</tbody>
</table>

**GetSurfaceTypeNameById**

Gets the surface type name by its identifier.

**Syntax**

```cpp
System.GetSurfaceTypeNameById(int surfaceId)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>surfaceId</td>
<td>Surface identifier.</td>
</tr>
</tbody>
</table>

**GetSystemMem**

Gets the amount of the memory for the system.

**Syntax**

```cpp
System.GetSystemMem()
```

**GetTerrainElevation**

Gets the terrain elevation of the specified position.

**Syntax**

```cpp
System.GetTerrainElevation(Vec3 v3Pos)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v3Pos</td>
<td>Position of the terraint to be checked.</td>
</tr>
</tbody>
</table>
**GetUserName**

Gets the username on this machine.

**Syntax**

```plaintext
System.GetUserName()
```

**GetViewCameraAngles**

Gets the view camera angles.

**Syntax**

```plaintext
System.GetViewCameraAngles()
```

**GetViewCameraDir**

Gets the view camera direction.

**Syntax**

```plaintext
System.GetViewCameraDir()
```

**GetViewCameraFov**

Gets the view camera fov.

**Syntax**

```plaintext
System.GetViewCameraFov()
```

**GetViewCameraPos**

Gets the view camera position.

**Syntax**

```plaintext
System.GetViewCameraPos()
```

**GetViewCameraUpDir**

Gets the view camera up-direction.

**Syntax**

```plaintext
System.GetViewCameraUpDir()
```

**GetWind**

Gets the wind direction.

**Syntax**

```plaintext
System.SetWind()
```
IsDevModeEnable
Checks if game is running in dev mode (cheat mode), which enables certain script function facilities (god mode, fly mode etc.).

Syntax

System.IsDevModeEnable()

IsEditing
Checks if the system is in pure editor mode - that is, not editor game mode.

Syntax

System.IsEditing()

IsEditor
Checks if the system is the editor.

Syntax

System.IsEditor()

IsHDRSupported
Checks if the HDR is supported.

Syntax

System.IsHDRSupported()

IsMultiplayer
Checks if the game is multiplayer.

Syntax

System.IsMultiplayer()

IsPointIndoors
Checks if a point is indoors.

Syntax

System.IsPointIndoors(Vec3 vPos)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vPos</td>
<td>Position vector. The default is (0, 0, 0).</td>
</tr>
</tbody>
</table>
IsPointVisible
Checks if the specified point is visible.

Syntax

System.IsPointVisible(Vec3 point)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>point</td>
<td>Point vector.</td>
</tr>
</tbody>
</table>

IsPS20Supported
Checks if the PS20 is supported.

Syntax

System.IsPS20Supported()

IsValidMapPos
Checks if the position is a valid map position.

Syntax

System.IsValidMapPos(Vec3 v)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>Position vector. The default is (0, 0, 0).</td>
</tr>
</tbody>
</table>

LoadFont
Loads a font.

Syntax

System.LoadFont(const char* pszName)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pszName</td>
<td>Font name.</td>
</tr>
</tbody>
</table>

LoadLocalizationXml
Loads Excel exported XML file with text and dialog localization data.
Syntax

System.LoadLocalizationXml(const char *filename)

Log
Logs a message to the log file and console.

Syntax

System.Log(const char* sText)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sText</td>
<td>Text to be logged.</td>
</tr>
</tbody>
</table>

LogAlways
Logs data even if the verbosity setting is 0.

Syntax

System.LogAlways(const char* sText)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sText</td>
<td>Text to be logged.</td>
</tr>
</tbody>
</table>

LogToConsole
Logs a message to the console.

Syntax

System.LogToConsole(const char* sText)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sText</td>
<td>Text to be logged.</td>
</tr>
</tbody>
</table>

PrepareEntityFromPool
Prepares the given bookmarked entity from the pool, bringing it into existence.

Syntax

System.PrepareEntityFromPool(entityId)
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>Entity identifier (number or ScriptHandle).</td>
</tr>
<tr>
<td>bPrepareNow</td>
<td>(optional) When another entity preparation is already in progress, specifies whether the pooled entity should be prepared immediately instead of putting it in a queue.</td>
</tr>
</tbody>
</table>

**ProjectToScreen**

Projects to the screen (not guaranteed to work if used outside Renderer).

**Syntax**

```c
System.ProjectToScreen(Vec3 vec)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vec</td>
<td>Position vector.</td>
</tr>
</tbody>
</table>

**Quit**

Quits the program.

**Syntax**

```c
System.Quit()
```

**QuitInNSeconds**

Quits the application in the specified number of seconds.

**Syntax**

```c
System.QuitInNSeconds(float fInNSeconds)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fInNSeconds</td>
<td>Number of seconds before quitting.</td>
</tr>
</tbody>
</table>

**RayTraceCheck**

Checks world and static objects.

**Syntax**
System.RayTraceCheck(Vec3 src, Vec3 dst, int skipId1, int skipId2)

**RayWorldIntersection**

Shoots rays into the world.

**Syntax**

System.RayWorldIntersection(Vec3 vPos, Vec3 vDir, int nMaxHits, int iEntTypes)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vPos</td>
<td>Position vector. The default is (0, 0, 0).</td>
</tr>
<tr>
<td>vDir</td>
<td>Direction vector. The default is (0, 0, 0).</td>
</tr>
<tr>
<td>nMaxHits</td>
<td>Maximum number of hits.</td>
</tr>
<tr>
<td>iEntTypes</td>
<td></td>
</tr>
</tbody>
</table>

**RemoveEntity**

Removes the specified entity.

**Syntax**

System.RemoveEntity(ScriptHandle entityId)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>Entity identifier.</td>
</tr>
</tbody>
</table>

**ResetPoolEntity**

Resets the entity's bookmarked, which frees memory.

**Syntax**

System.ResetPoolEntity(entityId)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>Entity identifier (svtnumber or ScriptHandle).</td>
</tr>
</tbody>
</table>

**ReturnEntityToPool**

**Syntax**
System.ReturnEntityToPool(entityId)

**Returns:** the bookmarked entity to the pool, destroying it.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>Entity identifier (svtnumber or ScriptHandle).</td>
</tr>
</tbody>
</table>

**SaveConfiguration**

Saves the configuration.

**Syntax**

```
System.SaveConfiguration()
```

**ScanDirectory**

Scans a directory.

**Syntax**

```
System.ScanDirectory(const char* pszFolderName, int nScanMode)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pszFolderName</td>
<td>Folder name.</td>
</tr>
<tr>
<td>nScanMode</td>
<td>Scan mode for the folder. Can be: SCANDIR_ALL (0), SCANDIR_FILES (1), or SCANDIR_SUBDIRS (2).</td>
</tr>
</tbody>
</table>

**ScreenToTexture**

**Syntax**

```
System.ScreenToTexture()
```

**SetBudget**

Sets system budget.

**Syntax**

```
System.SetBudget(int sysMemLimitInMB, int videoMemLimitInMB, float frameTimeLimitInMS, int soundChannelsPlayingLimit, int soundMemLimitInMB, int soundCPULimitInPercent, int numDrawCallsLimit)
```
## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysMemLimitInMB</td>
<td>Limit of the amount of system memory in MB. The default is 512.</td>
</tr>
<tr>
<td>videoMemLimitInMB</td>
<td>Limit of the amount of video memory in MB. The default is 256.</td>
</tr>
<tr>
<td>frameTimeLimitInMS</td>
<td>Limit of the frame time in MS. The default is 50.0f.</td>
</tr>
<tr>
<td>soundChannelsPlayingLimit</td>
<td>Limit of the number of sound channels playing. The default is 64.</td>
</tr>
<tr>
<td>soundMemLimitInMB</td>
<td>Limit of the sound memory in MB. The default is 64.</td>
</tr>
<tr>
<td>soundCPULimitInPercent</td>
<td>Limit of the sound CPU usage in percent. The default is 5.</td>
</tr>
<tr>
<td>numDrawCallsLimit</td>
<td>Limit of the number of draw calls. The default is 2000.</td>
</tr>
</tbody>
</table>

### SetConsoleImage

Sets the console image.

**Syntax**

```
System.SetConsoleImage(const char* pszName, bool bRemoveCurrent)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pszName</td>
<td>The name of the texture image.</td>
</tr>
<tr>
<td>bRemoveCurrent</td>
<td>True to remove the current image; otherwise false.</td>
</tr>
</tbody>
</table>

### SetCVar

Sets the value of a console variable.

**Syntax**

```
System.SetCVar(const char* sCVarName, value)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sCVarName</td>
<td>Name of the variable.</td>
</tr>
<tr>
<td>value</td>
<td>Value of the variable (float or string).</td>
</tr>
</tbody>
</table>

### SetGammaDelta

Sets the gamma/delta value.

**Syntax**
System.SetGammaDelta(float fDelta)

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fDelta</td>
<td>Delta value. The default is 0.</td>
</tr>
</tbody>
</table>

**SetOutdoorAmbientColor**

Sets the outdoor ambient color.

**Syntax**

```c
System.GetOutdoorAmbientColor(v3Color)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>v3Color</td>
<td>Outdoor ambient color value.</td>
</tr>
</tbody>
</table>

**SetPostProcessFxParam**

Sets a post processing effect parameter value.

**Syntax**

```c
System.SetPostProcessFxParam(const char* pszEffectParam, value)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pszEffectParam</td>
<td>Parameter for the post processing effect.</td>
</tr>
<tr>
<td>value</td>
<td>Value for the parameter (svtNumber, svtObject, or svtString).</td>
</tr>
</tbody>
</table>

**SetScissor**

Sets the scissoring screen area.

**Syntax**

```c
System.SetScissor(float x, float y, float w, float h)
```

**SetScreenFx**

Sets a post processing effect parameter value.

**Note**

This is a convenience wrapper function for `System.SetPostProcessFxParam`.  

---

910
Syntax

System.SetScreenFx(pszEffectParam, value)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pszEffectParam</td>
<td>Parameter for the post processing effect.</td>
</tr>
<tr>
<td>value</td>
<td>Value for the parameter (svtNumber, svtObject, or svtString).</td>
</tr>
</tbody>
</table>

SetSkyColor

Set color of the sky (outdoors ambient color).

Syntax

System.SetSkyColor(Vec3 vColor)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vColor</td>
<td>Sky Color as an {x, y, z} vector (x=r, y=g, z=b).</td>
</tr>
</tbody>
</table>

SetSkyHighlight

Set sky highlighting parameters.

Syntax

System.SetSkyHighlight(SmartScriptTable params)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>params</td>
<td>Table with sky highlighting parameters.</td>
</tr>
</tbody>
</table>

Params Table Parameters

<table>
<thead>
<tr>
<th>Highlight Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>Sky highlight color</td>
</tr>
<tr>
<td>direction</td>
<td>Direction of the sky highlight in world space.</td>
</tr>
<tr>
<td>pos</td>
<td>Position of the sky highlight in world space.</td>
</tr>
<tr>
<td>size</td>
<td>Sky highlight scale.</td>
</tr>
</tbody>
</table>
**SetSunColor**

Set the color of the sun, only relevant outdoors.

**Syntax**

```
System.SetSunColor(Vec3 vColor)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vColor</td>
<td>Sun color as an (x, y, z) vector ((x=r, y=g, z=b)).</td>
</tr>
</tbody>
</table>

**SetViewCameraFov**

Sets the view camera fov.

**Syntax**

```
System.SetViewCameraFov(float fov)
```

**SetVolumetricFogModifiers**

Sets the volumetric fog modifiers.

**Syntax**

```
System.SetVolumetricFogModifiers(float globalDensityModifier, float atmosphereHeightModifier)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>globalDensityModifier</td>
<td>Modifier for the global density.</td>
</tr>
<tr>
<td>atmosphereHeightModifier</td>
<td>Modifier for the atmosphere height.</td>
</tr>
</tbody>
</table>

**SetWaterVolumeOffset**

`SetWaterLevel` is not supported by the 3D engine for now.

**Syntax**

```
System.SetWaterVolumeOffset()
```

**SetWind**

Sets the wind direction.

**Syntax**
System.SetWind(Vec3 vWind)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vWind</td>
<td>Wind direction. The default value is (0, 0, 0).</td>
</tr>
</tbody>
</table>

ShowConsole

Shows or hides the console.

Syntax

System.ShowConsole(int nParam)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nParam</td>
<td>1 to show the console, 0 to hide. The default is 0.</td>
</tr>
</tbody>
</table>

ShowDebugger

Shows the debugger.

Syntax

System.ShowDebugger()

SpawnEntity

Spawns an entity.

Syntax

System.SpawnEntity(SmartScriptTable params)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>params</td>
<td>Entity parameters.</td>
</tr>
</tbody>
</table>

ViewDistanceGet

Gets the view distance.

Syntax

System.ViewDistanceSet()
**ViewDistanceSet**

Sets the view distance.

**Syntax**

```cpp
System.ViewDistanceSet(float fViewDist)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fViewDist</td>
<td>View distance.</td>
</tr>
</tbody>
</table>

**Warning**

Shows a message text with the warning severity.

**Syntax**

```cpp
System.Warning(const char* sParam)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sParam</td>
<td>The text to be logged. The default value is an empty string.</td>
</tr>
</tbody>
</table>

**ScriptBind Action Functions**

CryLua is deprecated and will be removed in a future version of Lumberyard.

Lists C++ action functions that can be called from Lua scripts.

**Topics**

- `ScriptBind_Action (p. 915)`
- `ScriptBind_ActionMapManager (p. 928)`
- `ScriptBind_ActorSystem (p. 930)`
- `ScriptBind_GameStatistics (p. 931)`
- `ScriptBind_GameToken (p. 933)`
- `ScriptBind_Inventory (p. 934)`
- `ScriptBind_ItemSystem (p. 936)`
- `ScriptBind_Network (p. 939)`
- `ScriptBind_UIAction (p. 939)`
- `ScriptBind_Vehicle (p. 951)`
- `ScriptBind_VehicleSeat (p. 958)`
- `ScriptBind_VehicleSystem (p. 959)`
ScriptBind_Action

CryLua is deprecated and will be removed in a future version of Lumberyard.

Lists the action related Lua script bind functions. When parameters are present, the data types indicated in the signatures reflect those of the underlying C++ function.

ActivateEffect
Activates the effect specified.

Syntax

Action.ActivateEffect(const char * name)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Specifies the effect to activate.</td>
</tr>
</tbody>
</table>

ActivateExtensionForGameObject
Activates a specified extension for a game object.

Syntax

Action.ActivateExtensionForGameObject(ScriptHandle entityId, const char *extension, bool activate)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The identifier of the entity.</td>
</tr>
<tr>
<td>extension</td>
<td>The name of the extension.</td>
</tr>
<tr>
<td>activate</td>
<td>Specify true to activate the extension or false to deactivate it.</td>
</tr>
</tbody>
</table>

AddAngleSignal
Adds an angle for the signal.

Syntax

Action.AddAngleSignal(ScriptHandle entityId, float fAngle, float fFlexibleBoundary, const char *sSignal)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The identifier of the entity.</td>
</tr>
<tr>
<td>fAngle</td>
<td>The angle value.</td>
</tr>
</tbody>
</table>
### Parameter | Description
---|---
fFlexibleBoundary | The size of the flexible boundary.
sSignal | The string for the signal.

### AddRangeSignal

Add a range for the signal.

**Syntax**

```
Action.AddRangeSignal(ScriptHandle entityId, float fRadius, float fFlexibleBoundary, const char *sSignal)
```

### Parameter | Description
---|---
entityId | The identifier of the entity.
fRadius | The radius of the range area.
fFlexibleBoundary | Flexible boundary size.
sSignal | String for signal.

### AddTargetRangeSignal

Add a target range signal that has the parameters specified.

**Syntax**

```
Action.AddTargetRangeSignal(ScriptHandle entityId, ScriptHandle targetId, float fRadius, float fFlexibleBoundary, const char *sSignal)
```

### Parameter | Description
---|---
entityId | The identifier of the entity.
targetId | The identifier of the target.
fRadius | The radius of the range area.
fFlexibleBoundary | The size of the flexible boundary.
sSignal | The string for the signal.

### BanPlayer

Bans a specified player.

**Syntax**

```
Action.BanPlayer(ScriptHandle entityId, const char* message)
```
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The identifier of the entity.</td>
</tr>
<tr>
<td>message</td>
<td>The message for the ban.</td>
</tr>
</tbody>
</table>

### BindGameObjectToNetwork

Binds a specified game object to the network.

**Syntax**

```
Action.BindGameObjectToNetwork(ScriptHandle entityId)
```

### CacheItemGeometry

Caches an item geometry.

**Syntax**

```
Action.CacheItemGeometry(const char *itemName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>itemName</td>
<td>The string name of the item.</td>
</tr>
</tbody>
</table>

### CacheItemSound

Caches an item sound.

**Syntax**

```
Action.CacheItemSound(const char *itemName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>itemName</td>
<td>The string name of the item.</td>
</tr>
</tbody>
</table>

### ClearEntityTags

Clears the tag for the specified entity.

**Syntax**

```
Action.ClearEntityTags(ScriptHandle entityId)
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The identifier of the entity.</td>
</tr>
</tbody>
</table>

**ClearStaticTag**

Clears the specified static tag for the specified entity.

**Syntax**

```c
Action.ClearStaticTag(ScriptHandle entityId, const char *staticId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The identifier of the entity.</td>
</tr>
<tr>
<td>staticId</td>
<td>The identifier of the static tag.</td>
</tr>
</tbody>
</table>

**ConnectToServer**

Connects to the server specified.

**Syntax**

```c
Action.ConnectToServer(char* server)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>String that specifies the server to connect to.</td>
</tr>
</tbody>
</table>

**CreateGameObjectForEntity**

Creates a game object for the entity ID specified.

**Syntax**

```c
Action.CreateGameObjectForEntity(ScriptHandle entityId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The identifier of the entity.</td>
</tr>
</tbody>
</table>

**DestroyRangeSignaling**

Removes range signaling.

**Syntax**

```c
Action.DestroyRangeSignaling(ScriptHandle entityId)
```
### ScriptBind Action Functions

#### DisableSignalTimer

Disables the signal timer.

**Syntax**

```
Action.DisableSignalTimer(ScriptHandle entityId, const char *sText)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The identifier of the entity.</td>
</tr>
<tr>
<td>sText</td>
<td>The text for the signal.</td>
</tr>
</tbody>
</table>

#### DontSyncPhysics

Instructs the engine to not synchronize physics for the specified entity.

**Syntax**

```
Action.DontSyncPhysics(ScriptHandle entityId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The identifier of the entity.</td>
</tr>
</tbody>
</table>

#### EnableRangeSignaling

Enables or disables range signaling for the specified entity.

**Syntax**

```
Action.EnableRangeSignaling(ScriptHandle entityId, bool bEnable)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The identifier of the entity.</td>
</tr>
<tr>
<td>bEnable</td>
<td>Enable or disable range signaling.</td>
</tr>
</tbody>
</table>

#### EnableSignalTimer

Enables the signal timer.

**Syntax**

```
Action.EnableSignalTimer(ScriptHandle entityId, bool bEnable)
```
**Action.EnableSignalTimer**

`Action.EnableSignalTimer(ScriptHandle entityId, const char *sText)`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The identifier of the entity.</td>
</tr>
<tr>
<td>sText</td>
<td>The text for the signal.</td>
</tr>
</tbody>
</table>

**ForceGameObjectUpdate**

Forces the game object to be updated.

**Syntax**

`Action.ForceGameObjectUpdate(ScriptHandle entityId, bool force)`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The identifier of the entity.</td>
</tr>
<tr>
<td>force</td>
<td>Specify <code>true</code> to force the update; specify <code>false</code> otherwise.</td>
</tr>
</tbody>
</table>

**GetClassName**

Returns the class name, if available, for specified `classId`.

**Syntax**

`Action.GetClassName(int classId)`

**GetPlayerList**

Retrieves the current players list.

**Syntax**

`Action.GetPlayerList()`

**GetServer**

Gets the server that corresponds to the number specified.

**Syntax**

`Action.GetServer(int number)`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>The number of the server.</td>
</tr>
</tbody>
</table>
GetServerTime

Gets the current time on the server.

Syntax

Action.GetServerTime()

GetWaterInfo

Gets information about the water at the position specified.

Syntax

Action.GetWaterInfo(Vec3 pos)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>The position for which information will be returned.</td>
</tr>
</tbody>
</table>

HasAI

Returns true if the entity has an AI object associated with it and has been registered with the AI System

Syntax

Action.HasAI(ScriptHandle entityId)

IsChannelOnHold

Checks if the channel specified is on hold.

Syntax

Action.IsChannelOnHold(int channelId)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelId</td>
<td>The identifier of the channel.</td>
</tr>
</tbody>
</table>

IsChannelSpecial

Returns true if the channel is special.

Syntax

Action.IsChannelSpecial()

IsClient

Returns true if the current script runs on a client.
**Syntax**

```
Action.IsClient()
```

**IsGameObjectProbablyVisible**

Returns `true` if the specified object is likely visible.

**Syntax**

```
Action.IsGameObjectProbablyVisible(ScriptHandle gameObject)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gameObject</td>
<td>The game object to check for likely visibility.</td>
</tr>
</tbody>
</table>

**IsGameStarted**

Returns `true` if the game has started.

**Syntax**

```
Action.IsGameStarted()
```

**IsImmersivenessEnabled**

Returns `true` if immersive multiplayer is enabled.

**Syntax**

```
Action.IsImmersivenessEnabled()
```

**IsRMIserver**

Returns `true` if the current script is running on an RMI (Remote Method Invocation) server.

**Syntax**

```
Action.IsRMIserver()
```

**IsServer**

Returns `true` if the current script runs on a server.

**Syntax**

```
Action.IsServer()
```

**LoadXML**

Loads XML data from the file specified. For more information, see [Using the Lua XML Loader](p. 731).

**Syntax**
Action.LoadXML(const char * definitionFile, const char * dataFile)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>definitionFile</td>
<td>Name of an XML file that declares the kind of data that is included in dataFile.</td>
</tr>
<tr>
<td>dataFile</td>
<td>The name of the XML file that contains the Lua data described in definitionFile.</td>
</tr>
</tbody>
</table>

**PauseGame**

Puts the game into pause mode.

**Syntax**

Action.PauseGame(bool pause)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pause</td>
<td>Specify true to set the game in pause mode. Specify false to resume the game.</td>
</tr>
</tbody>
</table>

**Persistent2DText**

Adds persistent 2D text.

**Syntax**

Action.Persistent2DText(const char* text, float size, Vec3 color, const char* name, float timeout)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>The text to be displayed.</td>
</tr>
<tr>
<td>size</td>
<td>The size of the 2D text.</td>
</tr>
<tr>
<td>color</td>
<td>The color of the 2D text.</td>
</tr>
<tr>
<td>name</td>
<td>The name assigned to the 2D text.</td>
</tr>
<tr>
<td>timeout</td>
<td>The timeout for the 2D text.</td>
</tr>
</tbody>
</table>

**PersistentArrow**

Adds a persistent arrow to the world.

**Syntax**

Action.PersistentArrow(Vec3 pos, float radius, Vec3 dir, Vec3 color, const char* name, float timeout)
### PersistentEntityTag

Adds a persistent entity tag.

**Syntax**

```c
Action.PersistentEntityTag(ScriptHandle entityId, const char *text)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The identifier of the entity.</td>
</tr>
<tr>
<td>text</td>
<td>The text for the entity tag.</td>
</tr>
</tbody>
</table>

### PersistentLine

Adds a persistent line to the world.

**Syntax**

```c
Action.PersistentLine(Vec3 start, Vec3 end, Vec3 color, const char* name, float timeout)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>The starting position of the line.</td>
</tr>
<tr>
<td>end</td>
<td>The ending position of the line.</td>
</tr>
<tr>
<td>color</td>
<td>The color of the line.</td>
</tr>
<tr>
<td>name</td>
<td>The name assigned to the line.</td>
</tr>
<tr>
<td>timeout</td>
<td>The timeout for the line.</td>
</tr>
</tbody>
</table>

### PersistentSphere

Adds a persistent sphere to the world.

**Syntax**

```c
Action.PersistentSphere(ScriptHandle entityId, Vec3 pos, Vec3 radius, Vec3 dir, Vec3 color, const char* name, float timeout)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>The position of the arrow.</td>
</tr>
<tr>
<td>radius</td>
<td>The radius of the arrow.</td>
</tr>
<tr>
<td>dir</td>
<td>The direction of the arrow.</td>
</tr>
<tr>
<td>color</td>
<td>The color of the arrow.</td>
</tr>
<tr>
<td>name</td>
<td>The name assigned to the arrow.</td>
</tr>
<tr>
<td>timeout</td>
<td>The timeout for the arrow.</td>
</tr>
</tbody>
</table>
Action.PersistentSphere(Vec3 pos, float radius, Vec3 color, const char* name, float timeout)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>The position of the sphere.</td>
</tr>
<tr>
<td>radius</td>
<td>The radius of the sphere.</td>
</tr>
<tr>
<td>color</td>
<td>The color of the sphere.</td>
</tr>
<tr>
<td>name</td>
<td>The name assigned to the sphere.</td>
</tr>
<tr>
<td>timeout</td>
<td>The timeout for the sphere.</td>
</tr>
</tbody>
</table>

PreLoadADB
Use this function to pre-cache ADB files.

Syntax

Action.PreLoadADB(const char* adbFileName)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adbFileName</td>
<td>The path and filename of the animation ADB file which is to be pre-loaded.</td>
</tr>
</tbody>
</table>

RefreshPings
Refreshes pings for all servers.

Syntax

Action.RefreshPings()

RegisterWithAI
Registers the entity to the AI system and creates an AI object associated with it.

Syntax

Action.RegisterWithAI()

ResetRangeSignaling
Resets range signaling.

Syntax

Action.ResetRangeSignaling(ScriptHandle entityId)
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The identifier of the entity.</td>
</tr>
</tbody>
</table>

**ResetSignalTimer**

Resets the rate for the signal timer.

**Syntax**

```c
Action.ResetSignalTimer(ScriptHandle entityId, const char *sText)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The identifier of the entity.</td>
</tr>
<tr>
<td>sText</td>
<td>The text for the signal.</td>
</tr>
</tbody>
</table>

**ResetToNormalCamera**

Resets the camera to the last valid view stored.

**Syntax**

```c
Action.ResetToNormalCamera()
```

**SaveXML**

Saves the specified XML data to the file system.

**Syntax**

```c
Action.SaveXML(const char * definitionFile, const char * dataFile, SmartScriptTable dataTable)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>definitionFile</td>
<td>Name of an XML file that declares the kind of data that is included in dataFile. For more information, see [Using the Lua XML Loader](p. 731).</td>
</tr>
<tr>
<td>dataFile</td>
<td>The name of the XML file that contains the Lua data described in definitionFile.</td>
</tr>
<tr>
<td>dataTable</td>
<td>The name of the data table.</td>
</tr>
</tbody>
</table>

**SendGameplayEvent**

Sends an event for the gameplay.

**Syntax**
Action.SendGameplayEvent(ScriptHandle entityId, int event)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The identifier of the entity.</td>
</tr>
<tr>
<td>event</td>
<td>The integer of the event.</td>
</tr>
</tbody>
</table>

**SetAimQueryMode**

Sets the aim query mode for the AI proxy. By default, the AI proxy queries the movement controller if the character is aiming. You can override this behavior by using a different keyword for the `mode` parameter.

**Syntax**

Action.SetAimQueryMode(ScriptHandle entityId, int mode)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The identifier of the entity.</td>
</tr>
<tr>
<td>mode</td>
<td>Specifies one of the following values: QueryAimFromMovementController (the default), OverriddenAndAiming, or OverriddenAndNotAiming</td>
</tr>
</tbody>
</table>

**SetNetworkParent**

Sets the network parent.

**Syntax**

Action.SetNetworkParent(ScriptHandle entityId, ScriptHandle parentID)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The identifier of the entity.</td>
</tr>
<tr>
<td>parentID</td>
<td>The identifier of the parent network.</td>
</tr>
</tbody>
</table>

**SetSignalTimerRate**

Sets the rate for the signal timer.

**Syntax**

Action.SetSignalTimerRate(ScriptHandle entityId, const char *sText, float fRateMin, float fRateMax)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The identifier of the entity.</td>
</tr>
</tbody>
</table>
### ScriptBind Action Functions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sText</td>
<td>The text for the signal.</td>
</tr>
<tr>
<td>fRateMin</td>
<td>The minimum rate for the signal timer.</td>
</tr>
<tr>
<td>fRateMax</td>
<td>The maximum rate for the signal timer.</td>
</tr>
</tbody>
</table>

#### SetViewCamera
Saves the previous valid view and overrides it with the current camera settings.

**Syntax**

```
Action.SetViewCamera()
```

#### ScriptBind_ActionMapManager

CryLua is deprecated in Lumberyard.

The action map manager provides a high-level interface to handle input controls inside a game. An action map is a set of key or button mappings for a particular game mode (such as controlling a helicopter). For more information, see Input in Amazon Lumberyard (p. 719).

#### EnableActionFilter
Enables or disables a specified action filter. An action filter allows actions like `moveleft` or `moveright` to succeed or fail. For more information, see Action Filters (p. 723).

**Syntax**

```
ActionMapManager.EnableActionFilter( name, enable )
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the filter.</td>
</tr>
<tr>
<td>enable</td>
<td>Specify true to enable the filter, or false to disable it.</td>
</tr>
</tbody>
</table>

#### EnableActionMap
Enables or disables an action map.

**Syntax**

```
ActionMapManager.EnableActionMap(const char *name, bool enable)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the action map to enable or disable.</td>
</tr>
</tbody>
</table>
### EnableActionMapManager

Enables or disables the action map manager.

**Syntax**

```csharp
ActionMapManager.EnableActionMapManager( enable, resetStateOnDisable )
```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>Specify true to enable the action map, or false to disable it.</td>
</tr>
<tr>
<td>resetStateOnDisable</td>
<td>Resets the action states when the action map manager is disabled.</td>
</tr>
</tbody>
</table>

### GetDefaultActionEntity

Retrieves the currently set default action entity.

**Syntax**

```csharp
ActionMapManager.GetDefaultActionEntity()```

### InitActionMaps

Initializes the action maps and filters found in the file specified.

**Syntax**

```csharp
ActionMapManager.InitActionMaps( path )```

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>The XML file path.</td>
</tr>
</tbody>
</table>

### IsFilterEnabled

Queries whether the filter specified is currently enabled.

**Syntax**

```csharp
ActionMapManager.IsFilterEnabled( filterName )```
### ScriptBind Action Functions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filterName</td>
<td>The name of the filter whose status to check.</td>
</tr>
</tbody>
</table>

#### LoadControllerLayoutFile

Loads the given controller layout into the action map manager.

**Syntax**

```c
ActionMapManager.LoadControllerLayoutFile( layoutName )
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>layoutName</td>
<td>The name of the layout.</td>
</tr>
</tbody>
</table>

#### LoadFromXML

Loads information from an XML file.

**Syntax**

```c
ActionMapManager.LoadFromXML(const char *name)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the XML file to load.</td>
</tr>
</tbody>
</table>

#### SetDefaultActionEntity

Sets a new default action entity. The action map manager assigns new action maps to the action entity that you set as the default.

**Syntax**

```c
ActionMapManager.SetDefaultActionEntity( id, updateAll )
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Specifies the EntityId of the action entity that is to become the default.</td>
</tr>
<tr>
<td>updateAll</td>
<td>Updates all existing action map assignments.</td>
</tr>
</tbody>
</table>

#### ScriptBind_ActorSystem

CryLua is deprecated and will be removed in a future version of Lumberyard.

---

930
Lists C++ actor system functions that can be called from Lua scripts.

**CreateActor**

Creates an actor.

**Syntax**

```cpp
ActorSystem.CreateActor(ScriptHandle channelId, SmartScriptTable actorParams)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelId</td>
<td>Identifier for the network channel.</td>
</tr>
<tr>
<td>actorParams</td>
<td>Parameters for the actor.</td>
</tr>
</tbody>
</table>

**ScriptBind_GameStatistics**

CryLua is deprecated and will be removed in a future version of Lumberyard.

Lists C++ game statistics functions that can be called from Lua script.

**AddGameElement**

Adds a game element to specified scope.

**Syntax**

```cpp
GameStatistics.AddGameElement(scopeID, elementID, locatorID, locatorValue [, table])
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scopeID</td>
<td>The identifier of the scope.</td>
</tr>
<tr>
<td>elementID</td>
<td>The identifier of the element to be added.</td>
</tr>
<tr>
<td>locatorID</td>
<td>The identifier of the locator.</td>
</tr>
<tr>
<td>locatorValue</td>
<td>The value of the locator.</td>
</tr>
<tr>
<td>table</td>
<td>Optional. The table of the element.</td>
</tr>
</tbody>
</table>

**BindTracker**

**Syntax**

```cpp
GameStatistics.BindTracker(name, tracker)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the tracker to bind.</td>
</tr>
</tbody>
</table>
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tracker</td>
<td>The IStatsTracker* to be bound.</td>
</tr>
</tbody>
</table>

#### CurrentScope

Returns the ID of current scope, or -1 if the stack is empty.

**Syntax**

```csharp
GameStatistics.CurrentScope()
```

#### Event

**Syntax**

```csharp
GameStatistics.Event()
```

#### PopGameScope

Removes the scope from the top of the stack.

**Syntax**

```csharp
GameStatistics.PopGameScope([checkScopeId])
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkScopeId</td>
<td>Optional. The identifier of the scope to be removed from the top of the stack.</td>
</tr>
</tbody>
</table>

#### PushGameScope

Pushes a scope on top of the stack.

**Syntax**

```csharp
GameStatistics.PushGameScope(scopeID)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scopeID</td>
<td>The identifier of the scope to be placed on top of the stack.</td>
</tr>
</tbody>
</table>

#### RemoveGameElement

Removes the element that has the supplied parameter values from the specified scope.

**Syntax**
GameStatistics.RemoveGameElement(scopeID, elementID, locatorID, locatorValue)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scopeID</td>
<td>The identifier of the scope.</td>
</tr>
<tr>
<td>elementID</td>
<td>The identifier of the element to be removed.</td>
</tr>
<tr>
<td>locatorID</td>
<td>The identifier of the locator.</td>
</tr>
<tr>
<td>locatorValue</td>
<td>The value of the locator.</td>
</tr>
</tbody>
</table>

StateValue

Syntax

GameStatistics.StateValue()

UnbindTracker

Syntax

GameStatistics.UnbindTracker(name, tracker)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the tracker to unbind.</td>
</tr>
<tr>
<td>tracker</td>
<td>The IStatsTracker* to unbind.</td>
</tr>
</tbody>
</table>

ScriptBind_GameToken

CryLua is deprecated and will be removed in a future version of Lumberyard.

Lists C++ game token functions that can be called from Lua script.

DumpAllTokens

Dump all game tokens with their values to the log.

Syntax

GameToken.DumpAllTokens()

GetToken

Gets the value of a game token.

Syntax
GameToken.GetToken(const char *sTokenName)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sTokenName</td>
<td>The name of the token whose value to get.</td>
</tr>
</tbody>
</table>

**SetToken**

Sets the value of a game token.

**Syntax**

GameToken.SetToken(const char* tokenName, any tokenValue)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tokenName</td>
<td>The name of the token.</td>
</tr>
<tr>
<td>tokenValue</td>
<td>The value to set.</td>
</tr>
</tbody>
</table>

**ScriptBind_Inventory**

CryLua is deprecated and will be removed in a future version of Lumberyard.

Lists C++ inventory management functions that you can call from Lua script.

**Clear**

Clears the inventory.

**Syntax**

Inventory.Clear()

**Destroy**

Destroys the inventory.

**Syntax**

Inventory.Destroy()

**Dump**

Dumps the inventory.

**Syntax**

Inventory.Dump()
GetAmmoCapacity
Gets the capacity for the specified ammunition.

Syntax

```cpp
Inventory.GetAmmoCapacity(const char *ammoName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ammoName</td>
<td>The name of the ammunition.</td>
</tr>
</tbody>
</table>

GetAmmoCount
Gets the amount of the specified ammunition name.

Syntax

```cpp
Inventory.GetAmmoCount(const char *ammoName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ammoName</td>
<td>The name of the ammunition.</td>
</tr>
</tbody>
</table>

GetCurrentItem
Gets the current item.

Syntax

```cpp
Inventory.GetCurrentItem()
```

GetCurrentItemId
Gets the identifier of the current item.

Syntax

```cpp
Inventory.GetCurrentItemId()
```

GetGrenadeWeaponByClass
Gets grenade weapon by class name.

Syntax

```cpp
Inventory.GetGrenadeWeaponByClass(const char *className)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>className</td>
<td>The name of the class.</td>
</tr>
</tbody>
</table>
GetItemByClass

Gets item by class name.

Syntax

```cpp
Inventory.GetItemByClass(const char *className)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>className</td>
<td>The name of the class.</td>
</tr>
</tbody>
</table>

HasAccessory

Checks if the inventory contains the specified accessory.

Syntax

```cpp
Inventory.HasAccessory(const char *accessoryName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accessoryName</td>
<td>The name of the accessory.</td>
</tr>
</tbody>
</table>

SetAmmoCount

Sets the amount of the specified ammunition.

Syntax

```cpp
Inventory.SetAmmoCount(const char *ammoName, int count)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ammoName</td>
<td>The name of the ammunition.</td>
</tr>
<tr>
<td>count</td>
<td>The count of the ammunition.</td>
</tr>
</tbody>
</table>

ScriptBind_ItemSystem

CryLua is deprecated and will be removed in a future version of Lumberyard.

Lists C++ functions for actor items and items in packs that you can call from Lua script.

GetPackItemByIndex

Gets a pack item from its index.

Syntax
### GetPackItemByIndex

Get the specified item by its index in the specified pack.

**Syntax**

```c
ItemSystem.GetPackItemByIndex(const char *packName, int index)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>packName</td>
<td>The name of the pack.</td>
</tr>
<tr>
<td>index</td>
<td>The index of the item to retrieve.</td>
</tr>
</tbody>
</table>

### GetPackNumItems

Get the number of items in the specified pack.

**Syntax**

```c
ItemSystem.GetPackNumItems(const char* packName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>packName</td>
<td>The name of the pack whose item count to retrieve.</td>
</tr>
</tbody>
</table>

### GetPackPrimaryItem

Get the primary item of the specified pack.

**Syntax**

```c
ItemSystem.GetPackPrimaryItem(const char *packName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>packName</td>
<td>The name of the pack whose primary item to retrieve.</td>
</tr>
</tbody>
</table>

### GiveItem

Give the specified item.

**Syntax**

```c
ItemSystem.GiveItem(const char *itemName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>itemName</td>
<td>The name of the item.</td>
</tr>
</tbody>
</table>

### GiveItemPack

Give the item pack specified to the actor specified.

```c
ItemSystem.GiveItemPack(const char *packName)
```
Syntax

ItemSystem.GiveItemPack(ScriptHandle actorId, const char *packName)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actorId</td>
<td>The actor identifier.</td>
</tr>
<tr>
<td>packName</td>
<td>The name of the pack.</td>
</tr>
</tbody>
</table>

Reset

Resets the item system.

Syntax

ItemSystem.Reset()

SerializePlayerLTLInfo

Serializes player level to level (LTL) information.

Syntax

ItemSystem.SerializePlayerLTLInfo(bool reading)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reading</td>
<td>Boolean value.</td>
</tr>
</tbody>
</table>

SetActorItem

Sets an actor item.

Syntax

ItemSystem.SetActorItem(ScriptHandle actorId, ScriptHandle itemId, bool keepHistory)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actorId</td>
<td>The actor identifier.</td>
</tr>
<tr>
<td>itemId</td>
<td>The item identifier.</td>
</tr>
<tr>
<td>keepHistory</td>
<td>True to keep the history; otherwise, false.</td>
</tr>
</tbody>
</table>

SetActorItemByName

Sets an actor item by name.
Syntax

```c
ItemSystem.SetActorItemByName(ScriptHandle actorId, const char *name, bool keepHistory)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actorId</td>
<td>The actor identifier.</td>
</tr>
<tr>
<td>name</td>
<td>The name of the actor item.</td>
</tr>
<tr>
<td>keepHistory</td>
<td>True to keep the history; otherwise, false.</td>
</tr>
</tbody>
</table>

**ScriptBind_Network**

CryLua is deprecated and will be removed in a future version of Lumberyard.

Lists C++ network functions that you can call from Lua script.

**Expose**

**Syntax**

```c
Network.Expose()
```

**ScriptBind_UIAction**

CryLua is deprecated and will be removed in a future version of Lumberyard.

**CallFunction**

Calls a function of the UI flash asset or the UIEventSystem.

**Syntax**

```c
UIAction.CallFunction(elementName, instanceID, functionName, [arg1], [arg2], [...])
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elementName</td>
<td>The UI element name as defined in the XML or UIEventSystem name as defined in a .cpp file.</td>
</tr>
<tr>
<td>instanceID</td>
<td>The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances. If used on UIEventSystem, no instance ID is ignored.</td>
</tr>
<tr>
<td>functionName</td>
<td>Function or event name.</td>
</tr>
<tr>
<td>args</td>
<td>List of arguments (optional).</td>
</tr>
</tbody>
</table>
**DisableAction**

Disables the UI Action.

**Syntax**

```lua
UIAction.DisableAction(actionName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionName</td>
<td>UI Action name.</td>
</tr>
</tbody>
</table>

**EnableAction**

Enables the UI Action.

**Syntax**

```lua
UIAction.EnableAction(actionName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionName</td>
<td>UI Action name.</td>
</tr>
</tbody>
</table>

**EndAction**

Ends a UI Action. This can be only used within a `UIAction` Lua script!

**Syntax**

```lua
UIAction.EndAction(table, disable, arguments)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>Must be <code>self</code>.</td>
</tr>
<tr>
<td>disable</td>
<td>If true, this action is disabled when it terminates.</td>
</tr>
<tr>
<td>arguments</td>
<td>The arguments to return from this action.</td>
</tr>
</tbody>
</table>

**GetAlpha**

Get move clip alpha value.

**Syntax**

```lua
UIAction.GetAlpha(elementName, instanceID, mcName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elementName</td>
<td>The UI element name as defined in the XML.</td>
</tr>
</tbody>
</table>
### GetArray

Returns a table with values of the array.

**Syntax**

```plaintext
UIAction.GetArray(elementName, instanceID, arrayName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>elementName</code></td>
<td>The UI element name as defined in the XML.</td>
</tr>
<tr>
<td><code>instanceID</code></td>
<td>The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances.</td>
</tr>
<tr>
<td><code>arrayName</code></td>
<td>Array name as defined in the XML.</td>
</tr>
</tbody>
</table>

### GetPos

Get movie clip position.

**Syntax**

```plaintext
UIAction.GetPos(elementName, instanceID, mcName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>elementName</code></td>
<td>The UI element name as defined in the XML.</td>
</tr>
<tr>
<td><code>instanceID</code></td>
<td>The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances.</td>
</tr>
<tr>
<td><code>mcName</code></td>
<td>The movie clip name as defined in the XML.</td>
</tr>
</tbody>
</table>

### GetRotation

Get movie clip rotation.

**Syntax**

```plaintext
UIAction.GetRotation(elementName, instanceID, mcName)
```
### GetScale

Get movie clip scale.

**Syntax**

```javascript
UIAction.GetScale(elementName, instanceID, mcName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elementName</td>
<td>The UI element name as defined in the XML.</td>
</tr>
<tr>
<td>instanceID</td>
<td>The ID of the instance (if an instance with the specified ID does not exist, it will be created). '-1' for all instances.</td>
</tr>
<tr>
<td>mcName</td>
<td>The movie clip name as defined in the XML.</td>
</tr>
</tbody>
</table>

### GetVariable

Gets a variable of the UI flash asset.

**Syntax**

```javascript
UIAction.GetVariable(elementName, instanceID, varName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elementName</td>
<td>The UI element name as defined in the XML.</td>
</tr>
<tr>
<td>instanceID</td>
<td>The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances.</td>
</tr>
<tr>
<td>varName</td>
<td>Variable name as defined in the XML.</td>
</tr>
</tbody>
</table>

### GotoAndPlay

Call `GotoAndPlay` on a movie clip.

**Syntax**

```javascript
GotoAndPlay
```
UIAction.GotoAndPlay(elementName, instanceID, mcName, frameNum)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elementName</td>
<td>The UI element name as defined in the XML.</td>
</tr>
<tr>
<td>instanceID</td>
<td>The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances.</td>
</tr>
<tr>
<td>mcName</td>
<td>The movie clip name as defined in the XML.</td>
</tr>
<tr>
<td>frameNum</td>
<td>The frame number.</td>
</tr>
</tbody>
</table>

**GotoAndPlayFrameName**

Call GotoAndPlay on a movie clip by frame name.

**Syntax**

UIAction.GotoAndPlayFrameName(elementName, instanceID, mcName, frameName)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elementName</td>
<td>The UI element name as defined in the XML.</td>
</tr>
<tr>
<td>instanceID</td>
<td>The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances.</td>
</tr>
<tr>
<td>mcName</td>
<td>The movie clip name as defined in the XML.</td>
</tr>
<tr>
<td>frameName</td>
<td>The name of the frame.</td>
</tr>
</tbody>
</table>

**GotoAndStop**

Call GotoAndStop on a movie clip.

**Syntax**

UIAction.GotoAndStop(elementName, instanceID, mcName, frameNum)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elementName</td>
<td>The UI element name as defined in the XML.</td>
</tr>
<tr>
<td>instanceID</td>
<td>The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances.</td>
</tr>
<tr>
<td>mcName</td>
<td>The movie clip name as defined in the XML.</td>
</tr>
<tr>
<td>frameNum</td>
<td>The frame number.</td>
</tr>
</tbody>
</table>
### GotoAndStopFrameName

Call `GotoAndStop` on a movie clip by frame name.

**Syntax**

```
UIAction.GotoAndStopFrameName(elementName, instanceID, mcName, frameName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elementName</td>
<td>The UI element name as defined in the XML.</td>
</tr>
<tr>
<td>instanceID</td>
<td>The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances.</td>
</tr>
<tr>
<td>mcName</td>
<td>The movie clip name as defined in the XML.</td>
</tr>
<tr>
<td>frameName</td>
<td>The name of the frame.</td>
</tr>
</tbody>
</table>

### HideElement

Hide the UI flash asset.

**Syntax**

```
UIAction.HideElement(elementName, instanceID)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elementName</td>
<td>The UI element name as defined in the XML.</td>
</tr>
<tr>
<td>instanceID</td>
<td>The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances.</td>
</tr>
</tbody>
</table>

### IsVisible

Get movie clip visible state.

**Syntax**

```
UIAction.IsVisible(elementName, instanceID, mcName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elementName</td>
<td>The UI element name as defined in the XML.</td>
</tr>
<tr>
<td>instanceID</td>
<td>The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances.</td>
</tr>
</tbody>
</table>
### Parameter | Description
--- | ---
mCName | The movie clip name as defined in the XML.

**RegisterActionListener**

Register a callback function for a UIAction event. The callback function must have form:

CallbackName(actionName, eventName, argTable)

**Syntax**

```plaintext
UIAction.RegisterActionListener(table, actionName, eventName, callbackFunctionName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>The script that receives the callback (can be <code>self</code> to refer the current script).</td>
</tr>
<tr>
<td>actionName</td>
<td>The UI action name.</td>
</tr>
<tr>
<td>eventName</td>
<td>The name of the event that is fired from the UI action (can be <code>OnStart</code> or <code>OnEnd</code>)</td>
</tr>
<tr>
<td></td>
<td><strong>Warning</strong> If an empty string is specified, all events will be received.</td>
</tr>
<tr>
<td>callbackFunctionName</td>
<td>The name of the script function that will receive the callback.</td>
</tr>
</tbody>
</table>

**RegisterElementListener**

Register a callback function for a UIElement event. The callback function must have form:

CallbackName(elementName, instanceId, eventName, argTable)

**Syntax**

```plaintext
UIAction.RegisterElementListener(table, elementName, instanceId, eventName, callbackFunctionName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>The script that receives the callback (can be <code>self</code> to refer the current script).</td>
</tr>
<tr>
<td>elementName</td>
<td>The UI element name as defined in the XML.</td>
</tr>
<tr>
<td>instanceId</td>
<td>The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances.</td>
</tr>
<tr>
<td>eventName</td>
<td>The name of the event that is fired from the UI element. If an empty string is specified, all events will be received.</td>
</tr>
</tbody>
</table>
### RegisterEventSystemListener

Register a callback function for a UIEventSystem event. The callback function must have form: `CallbackName(actionName, eventName, argTable)`

**Syntax**

```lua
UIAction.RegisterEventSystemListener(table, eventSystem, eventName, callbackFunctionName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>callbackFunctionName</td>
<td>name of the script function that will receive the callback.</td>
</tr>
<tr>
<td>eventSystem</td>
<td>The UI event system name.</td>
</tr>
<tr>
<td>eventName</td>
<td>The name of the event that is fired from UIEventSystem. If an empty string is specified, all events will be received.</td>
</tr>
<tr>
<td>callbackFunctionName</td>
<td>name of the script function that will receive the callback.</td>
</tr>
</tbody>
</table>

### ReloadElement

Reloads the UI flash asset.

**Syntax**

```lua
UIAction.ReloadElement(elementName, instanceID)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elementName</td>
<td>The UI element name as defined in the XML.</td>
</tr>
<tr>
<td>instanceID</td>
<td>The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances.</td>
</tr>
</tbody>
</table>

### RequestHide

Send the fade out signal to the UI flash asset.

**Syntax**

```lua
UIAction.RequestHide(elementName, instanceID)
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elementName</td>
<td>The UI element name as defined in the XML.</td>
</tr>
<tr>
<td>instanceID</td>
<td>The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances.</td>
</tr>
</tbody>
</table>

**SetAlpha**

Set movie clip alpha value.

**Syntax**

```javascript
UIAction.SetAlpha(elementName, instanceID, mcName, fAlpha)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elementName</td>
<td>The UI element name as defined in the XML.</td>
</tr>
<tr>
<td>instanceID</td>
<td>The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances.</td>
</tr>
<tr>
<td>mcName</td>
<td>The movie clip name as defined in the XML.</td>
</tr>
<tr>
<td>fAlpha</td>
<td>Alpha value (0-1).</td>
</tr>
</tbody>
</table>

**SetArray**

Sets an array of the UI flash asset.

**Syntax**

```javascript
UIAction.SetArray(elementName, instanceID, arrayName, values)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elementName</td>
<td>The UI element name as defined in the XML.</td>
</tr>
<tr>
<td>instanceID</td>
<td>The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances.</td>
</tr>
<tr>
<td>arrayName</td>
<td>The array name as defined in the XML.</td>
</tr>
<tr>
<td>values</td>
<td>Table of values for the array.</td>
</tr>
</tbody>
</table>

**SetPos**

Set movie clip position.
**Syntax**

```plaintext
UIAction.SetPos(elementName, instanceID, mcName, vPos)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elementName</td>
<td>The UI element name as defined in the XML.</td>
</tr>
<tr>
<td>instanceID</td>
<td>The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances.</td>
</tr>
<tr>
<td>mcName</td>
<td>The movie clip name as defined in the XML.</td>
</tr>
<tr>
<td>vPos</td>
<td>position.</td>
</tr>
</tbody>
</table>

**SetRotation**

Set movie clip rotation.

**Syntax**

```plaintext
UIAction.SetRotation(elementName, instanceID, mcName, vRotation)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elementName</td>
<td>The UI element name as defined in the XML.</td>
</tr>
<tr>
<td>instanceID</td>
<td>The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances.</td>
</tr>
<tr>
<td>mcName</td>
<td>The movie clip name as defined in the XML.</td>
</tr>
<tr>
<td>vRotation</td>
<td>The rotation.</td>
</tr>
</tbody>
</table>

**SetScale**

Set movie clip scale.

**Syntax**

```plaintext
UIAction.SetScale(elementName, instanceID, mcName, vScale)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elementName</td>
<td>The UI element name as defined in the XML.</td>
</tr>
<tr>
<td>instanceID</td>
<td>The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances.</td>
</tr>
<tr>
<td>mcName</td>
<td>The movie clip name as defined in the XML.</td>
</tr>
</tbody>
</table>
### Parameter | Description
---|---
vScale | scale.

### SetVariable
Sets a variable of the UI flash asset.

**Syntax**

```plaintext
UIAction.SetVariable(elementName, instanceID, varName, value)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
elementName | The UI element name as defined in the XML. |
instanceID | The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances. |
varName | Variable name as defined in the XML. |
value | Value to set. |

### SetVisible
Set movie clip visible state.

**Syntax**

```plaintext
UIAction.SetVisible(elementName, instanceID, mcName, bVisible)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
elementName | The UI element name as defined in the XML. |
instanceID | The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances. |
mcName | The movie clip name as defined in the XML. |
bVisible | visible. |

### ShowElement
Displays the UI flash asset.

**Syntax**

```plaintext
UIAction.ShowElement(elementName, instanceID)
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elementName</td>
<td>The UI element name as defined in the XML.</td>
</tr>
<tr>
<td>instanceID</td>
<td>The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances.</td>
</tr>
</tbody>
</table>

**StartAction**

Starts a UI Action.

**Syntax**

```
UIAction.StartAction(actionName, arguments)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionName</td>
<td>UI Action name.</td>
</tr>
<tr>
<td>arguments</td>
<td>The arguments to pass to this action.</td>
</tr>
</tbody>
</table>

**UnloadElement**

Unloads the UI flash asset.

**Syntax**

```
UIAction.UnloadElement(elementName, instanceID)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elementName</td>
<td>The UI element name as defined in the XML.</td>
</tr>
<tr>
<td>instanceID</td>
<td>The ID of the instance (if an instance with the specified ID does not exist, it will be created). -1 specifies all instances.</td>
</tr>
</tbody>
</table>

**UnregisterActionListener**

Unregister callback functions for a UIAction event.

**Syntax**

```
UIAction.UnregisterActionListener(table, callbackFunctionName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>The script that receives the callback (can be self to refer the current script).</td>
</tr>
</tbody>
</table>
### Parameter | Description
--- | ---
callbackFunctionName | The name of the script function that receives the callback. If "" is specified, all callbacks for this script will be removed.

#### UnregisterElementListener
Unregister callback functions for a UIElement event.

**Syntax**

```
UIAction.UnregisterElementListener(table, callbackFunctionName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>The script that receives the callback (can be self to refer the current script).</td>
</tr>
<tr>
<td>callbackFunctionName</td>
<td>The name of the script function that receives the callback. If &quot;&quot; is specified, all callbacks for this script will be removed.</td>
</tr>
</tbody>
</table>

#### UnregisterEventSystemListener
Unregister callback functions for a UIEventSystem event.

**Syntax**

```
UIAction.UnregisterEventSystemListener(table, callbackFunctionName)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>The script that receives the callback (can be self to refer the current script).</td>
</tr>
<tr>
<td>callbackFunctionName</td>
<td>The name of the script function that receives the callback. If &quot;&quot; is specified, all callbacks for this script will be removed.</td>
</tr>
</tbody>
</table>

#### ScriptBind_Vehicle

CryLua is deprecated and will be removed in a future version of Lumberyard.

Lists C++ vehicle system functions that you can call from Lua script.

#### AddSeat

Adds a seat to the vehicle.
Syntax

```
Vehicle.AddSeat(SmartScriptTable paramsTable)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>paramsTable</td>
<td>The seat parameters in SmartScriptTable format.</td>
</tr>
</tbody>
</table>

**ChangeSeat**

Makes the actor change the seat inside the vehicle.

Syntax

```
Vehicle.ChangeSeat(ScriptHandle actorHandle, int seatId, bool isAnimationEnabled)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actorHandle</td>
<td>The actor identifier.</td>
</tr>
<tr>
<td>seatId</td>
<td>The seat identifier.</td>
</tr>
<tr>
<td>isAnimationEnabled</td>
<td>True if animation is enabled; otherwise, false.</td>
</tr>
</tbody>
</table>

**Destroy**

Destroys the vehicle.

Syntax

```
Vehicle.Destroy()
```

**DisableEngine**

Disables or enables the engine of the vehicle.

Syntax

```
Vehicle.DisableEngine(bool disable)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disable</td>
<td>True to disable the engine; false to enable.</td>
</tr>
</tbody>
</table>

**EnableMovement**

Enables or disables the movement of the vehicle.

Syntax
Vehicle.EnableMovement(bool enable)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>True to enable movement; false to disable.</td>
</tr>
</tbody>
</table>

**EnterVehicle**

Makes the specified actor enter the vehicle.

**Syntax**

Vehicle.EnterVehicle(ScriptHandle actorHandle, int seatId, bool isAnimationEnabled)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actorHandle</td>
<td>The actor identifier.</td>
</tr>
<tr>
<td>seatId</td>
<td>The seat identifier.</td>
</tr>
<tr>
<td>isAnimationEnabled</td>
<td>True if animation is enabled; otherwise, false.</td>
</tr>
</tbody>
</table>

**ExitVehicle**

Makes the actor leave the vehicle.

**Syntax**

Vehicle.ExitVehicle(ScriptHandle actorHandle)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actorHandle</td>
<td>The actor identifier.</td>
</tr>
</tbody>
</table>

**GetComponentDamageRatio**

Gets the damage ratio of the specified component.

**Syntax**

Vehicle.GetComponentDamageRatio(const char* pComponentName)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pComponentName</td>
<td>The name of the component.</td>
</tr>
</tbody>
</table>

**GetHelperDir**

Gets the helper direction.
### Syntax

```
Vehicle.GetHelperDir(const char* name, bool isInVehicleSpace)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the helper.</td>
</tr>
<tr>
<td>isInVehicleSpace</td>
<td>True if the helper is in the vehicle space; otherwise, false.</td>
</tr>
</tbody>
</table>

**GetHelperPos**

Gets the helper position.

### Syntax

```
Vehicle.GetHelperPos(const char* name, bool isInVehicleSpace)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the helper.</td>
</tr>
<tr>
<td>isInVehicleSpace</td>
<td>True if the helper is in the vehicle space; otherwise, false.</td>
</tr>
</tbody>
</table>

**GetHelperWorldPos**

Gets the helper position in the world coordinates.

### Syntax

```
Vehicle.GetHelperWorldPos(const char* name)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the helper.</td>
</tr>
</tbody>
</table>

**GetSeatForPassenger**

Returns a vehicle seat ID for the specified passenger.

### Syntax

```
Vehicle.GetSeatForPassenger(ScriptHandle passengerId)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>passengerId</td>
<td>The passenger ID.</td>
</tr>
</tbody>
</table>
GetVehicle

Gets the vehicle identifier.

Syntax

Vehicle.GetVehicle()

HasHelper

Checks if the vehicle has the specified helper.

Syntax

Vehicle.HasHelper(const char* name)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the helper.</td>
</tr>
</tbody>
</table>

IsDestroyed

Checks if the vehicle is destroyed.

Syntax

Vehicle.IsDestroyed()

IsInsideRadius

Checks if the vehicle is inside the specified radius.

Syntax

Vehicle.IsInsideRadius(Vec3 pos, float radius)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>The ((x,y,z)) position vector.</td>
</tr>
<tr>
<td>radius</td>
<td>The radius, expressed as a float.</td>
</tr>
</tbody>
</table>

IsUsable

Checks if the vehicle is usable by the user.

Syntax

Vehicle.IsUsable(ScriptHandle userHandle)
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userHandle</td>
<td>The user identifier.</td>
</tr>
</tbody>
</table>

**MultiplyWithWorldTM**
Multiplying with the world transformation matrix.

**Syntax**
```
Vehicle.MultiplyWithWorldTM(Vec3 pos)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos</td>
<td>The ( {x,y,z} ) position vector.</td>
</tr>
</tbody>
</table>

**OnHit**
Triggers an event that occurs after the vehicle is hit.

**Syntax**
```
Vehicle.OnHit(ScriptHandle targetId, ScriptHandle shooterId, float damage, Vec3 position, float radius, int hitTypeId, bool explosion)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetId</td>
<td>The target identifier.</td>
</tr>
<tr>
<td>shooterId</td>
<td>The shooter identifier.</td>
</tr>
<tr>
<td>damage</td>
<td>The amount of damage, expressed as a float.</td>
</tr>
<tr>
<td>position</td>
<td>The ( {x,y,z} ) position vector.</td>
</tr>
<tr>
<td>radius</td>
<td>Radius of the hit, expressed as a float.</td>
</tr>
<tr>
<td>hitTypeId</td>
<td>The type of damage, expressed as an integer.</td>
</tr>
<tr>
<td>explosion</td>
<td>True if the hit causes an explosion, otherwise false.</td>
</tr>
</tbody>
</table>

**OnSpawnComplete**
Calls back into the game code for when vehicle spawn has been completed.

**Syntax**
```
Vehicle.OnSpawnComplete()
```

**OnUsed**
Triggers an event when the user uses the specified vehicle.
ScriptBind Action Functions

Syntax

Vehicle.OnUsed(ScriptHandle userHandle, int index)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userHandle</td>
<td>The user identifier.</td>
</tr>
<tr>
<td>index</td>
<td>The seat identifier.</td>
</tr>
</tbody>
</table>

ProcessPassengerDamage

Processes passenger damages.

Syntax

Vehicle.ProcessPassengerDamage(ScriptHandle passengerId, float actorHealth, float damage, int hitTypeId, bool explosion)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>passengerId</td>
<td>The passenger identifier.</td>
</tr>
<tr>
<td>actorHealth</td>
<td>The health of the actor.</td>
</tr>
<tr>
<td>damage</td>
<td>The amount of damage.</td>
</tr>
<tr>
<td>hitTypeId</td>
<td>The type of damage.</td>
</tr>
<tr>
<td>explosion</td>
<td>True if there is an explosion; otherwise, false.</td>
</tr>
</tbody>
</table>

Reset

Resets the vehicle.

Syntax

Vehicle.Reset()

ResetSlotGeometry

Syntax

Vehicle.ResetSlotGeometry(int slot, const char* filename, const char* geometry)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slot</td>
<td>The number of the slot.</td>
</tr>
<tr>
<td>filename</td>
<td>The filename.</td>
</tr>
<tr>
<td>geometry</td>
<td>The slot geometry.</td>
</tr>
</tbody>
</table>
**ScriptBind_VehicleSeat**

CryLua is deprecated and will be removed in a future version of Lumberyard.

Lists C++ vehicle seat functions that you can call from Lua script.

**GetPassengerId**

Gets the passenger identifier.

**Syntax**

```
VehicleSeat.GetPassengerId()
```

**GetVehicleSeat**

Gets the vehicle seat identifier.

**Syntax**

```
VehicleSeat.GetVehicleSeat()
```

**GetWeaponCount**

Gets the number of weapons available on this seat.

**Syntax**

```
VehicleSeat.GetWeaponCount()
```

**GetWeaponId**

Gets the weapon identifier.

**Syntax**

```
VehicleSeat.GetWeaponId(int weaponIndex)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>weaponIndex</td>
<td>Weapon identifier.</td>
</tr>
</tbody>
</table>

**IsDriver**

Checks if the seat is the driver seat.

**Syntax**

```
VehicleSeat.IsDriver()
```
IsFree
Checks if the seat is free.

Syntax

VehicleSeat.IsFree(ScriptHandle actorHandle)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actorHandle</td>
<td>Passenger identifier.</td>
</tr>
</tbody>
</table>

IsGunner
Checks if the seat is the gunner seat.

Syntax

VehicleSeat.IsGunner()

Reset
Resets the vehicle seat.

Syntax

VehicleSeat.Reset()

SetAIWeapon
Sets the weapon artificial intelligence.

Syntax

VehicleSeat.SetAIWeapon(ScriptHandle weaponHandle)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>weaponHandle</td>
<td>Weapon identifier.</td>
</tr>
</tbody>
</table>

ScriptBind_VehicleSystem

CryLua is deprecated and will be removed in a future version of Lumberyard.

Lists C++ vehicle system functions that you can call from Lua script.

GetOptionalScript
Get an (optional) script for the named vehicle.
Syntax

VehicleSystem.GetOptionalScript(char* vehicleName)

**GetVehicleImplementations**

Get a table of all implemented vehicles.

**Syntax**

VehicleSystem.GetVehicleImplementations()

**ReloadSystem**

Reloads the vehicle system with default values.

**Syntax**

VehicleSystem.ReloadSystem()

**SetTpvDistance**

Distance of camera in third person view.

**Syntax**

VehicleSystem.SetTpvDistance(float distance)

**SetTpvHeight**

Height of camera in third person view.

**Syntax**

VehicleSystem.SetTpvHeight(float height)

**ScriptBind_Boids**

CryLua is deprecated and will be removed in a future version of Lumberyard.

These functions create simulated flocks of bird-like objects (boids) or other animals and control their behavior.

**CanPickup**

**Syntax**

Checks if the boid can be picked up.

Boids.CanPickup(flockEntity, boidEntity)
### CreateBugsFlock

Creates a bugs flock and binds it to the given entity.

**Syntax**

```cpp
Boids.CreateBugsFlock(entity, paramsTable)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>Valid entity table.</td>
</tr>
<tr>
<td>paramTable</td>
<td>Table with parameters for flock (see sample scripts).</td>
</tr>
</tbody>
</table>

### CreateFishFlock

Creates a fish flock and binds it to the given entity.

**Syntax**

```cpp
Boids.CreateFishFlock(entity, paramsTable)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>Valid entity table.</td>
</tr>
<tr>
<td>paramTable</td>
<td>Table with parameters for flock (see sample scripts).</td>
</tr>
</tbody>
</table>

### CreateFlock

Creates a flock of boids and binds it to the given entity.

**Syntax**

```cpp
Boids.CreateFlock(entity, paramsTable)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>Valid entity table.</td>
</tr>
<tr>
<td>nType</td>
<td>The type of flock. Possible values are <code>Boids.FLOCK_BIRDS</code>, <code>Boids.FLOCK_FISH</code>, or <code>Boids.FLOCK_BUGS</code>.</td>
</tr>
</tbody>
</table>
### EnableFlock

Enables or disables a flock in the entity.

**Syntax**

```plaintext
Boids.EnableFlock(entity, paramsTable)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>Valid entity table containing flock.</td>
</tr>
<tr>
<td>bEnable</td>
<td>Specify true to enable the flock; false to disable.</td>
</tr>
</tbody>
</table>

### GetUsableMessage

Gets the appropriate localized UI message for the specified flock.

**Syntax**

```plaintext
Boids.GetUsableMessage(flockEntity)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flockEntity</td>
<td>Valid entity table containing flock.</td>
</tr>
</tbody>
</table>

### OnBoidHit

Event that occurs on boid hit.

**Syntax**

```plaintext
Boids.OnBoidHit(flockEntity, boidEntity, hit)
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flockEntity</td>
<td>Valid entity table containing flock.</td>
</tr>
<tr>
<td>boidEntity</td>
<td>Valid entity table containing boid.</td>
</tr>
<tr>
<td>hit</td>
<td>Valid entity table containing hit information.</td>
</tr>
</tbody>
</table>

### OnPickup

Forwards the appropriate pickup action to the boid object.
Syntax

Boids.OnPickup(flockEntity, boidEntity, bPickup, fThrowSpeed)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flockEntity</td>
<td>Valid entity table containing flock.</td>
</tr>
<tr>
<td>boidEntity</td>
<td>Valid entity table containing boid.</td>
</tr>
<tr>
<td>bPickup</td>
<td>Pickup, or drop or throw.</td>
</tr>
<tr>
<td>fThrowSpeed</td>
<td>Specifies the throw speed. By default, a value greater than 5.f kills the boid. This has no effect on the pickup action.</td>
</tr>
</tbody>
</table>

SetAttractionPoint

Sets the one time attraction point for the boids.

Syntax

Boids.SetAttractionPoint(entity, paramsTable)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>Valid entity table containing flock.</td>
</tr>
<tr>
<td>point</td>
<td>The one time attraction point.</td>
</tr>
</tbody>
</table>

SetFlockParams

Sets the parameters of the flock for the specified entity.

Syntax

Boids.SetFlockParams(entity, paramsTable)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>Valid entity table containing flock.</td>
</tr>
<tr>
<td>paramTable</td>
<td>Table with parameters for flock (see sample scripts).</td>
</tr>
</tbody>
</table>

SetFlockPercentEnabled

Specifies the percentage of boid objects that are rendered in flocks. You can use this to enable flocks gradually.

Syntax
Boids.SetFlockPercentEnabled(entity, paramsTable)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>Valid entity table containing flock.</td>
</tr>
<tr>
<td>nPercent</td>
<td>Possible values are from 0 through 100. If 0, no boids are rendered; if 100, all boids are rendered.</td>
</tr>
</tbody>
</table>

**Integrating Lua and C++**

CryLua is deprecated and will be removed in a future version of Lumberyard.

The CryScript system abstracts a Lua virtual machine for use by the other systems and the game code. It includes the following functionality:

- calling script functions
- exposing C++-based variables and functions to scripts
- creating script tables stored in virtual machine memory

The CryScript system is based on Lua 5. More information on the Lua language can be found at [http://www.lua.org](http://www.lua.org).

**Accessing Script Tables**

A global script table can be retrieved by calling IScriptSystem::GetGlobalValue(). The IScriptTable is used to represent all script tables/variables.

**Exposing C++ Functions and Values**

To expose C++ functions and variables to scripts, you'll need to implement a new class. The easiest way is to derive the CScriptableBase class, which provides most of the functionality.

**Exposing Constants**

To expose constant values to scripts, use the IScriptSystem::SetGlobalValue(). For example, to expose a constant named MTL_LAYER_FROZEN to our scripts, use the following code:

```cpp
 Nit material::pScriptSystem->SetGlobalValue("MTL_LAYER_FROZEN", MTL_LAYER_FROZEN);
```

**Exposing Functions**

To expose C++ functions to scripts, implement a new class derives from CScriptableBase, as shown in the following example.

```cpp
class CScriptBind_Game : public CScriptableBase {
```

964
public:
    CScriptBind_Game(ISystem* pSystem);
    virtual ~CScriptBind_Game() {}

    intGameLog(IFunctionHandler* pH, char* pText);
};

Add the following code inside the class constructor:

Init(pSystem->GetIScriptSystem(), pSystem);
SetGlobalName("Game");
#undef SCRIPT_REG_CLASSNAME
#define SCRIPT_REG_CLASSNAME &CScriptBind_Game::

SCRIPT_REG_TEMPLFUNC(GameLog, "text");

In a Lua script, you can access your new ScriptBind function as follows:

Game.GameLog("a message")
Object and Entity System

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 about the latest features, see the Amazon Lumberyard User Guide.

Using the Object and Entity system, you can create and place objects, brushes, and entities in your level. Entities are objects with which the player interacts. Similar to brushes, they can be placed in a level, and are accessed from the Objects tab of the Rollup Bar.

Note
The component entity system replaces the existing entity system in Lumberyard at a future date.

Topics
- Using the Designer Tool (p. 967)
- Using the Measurement System Tool (p. 980)
- Using the Object Selector (p. 980)
- Brushes (p. 983)
- Prefabs (p. 985)
- Common Parameters and Properties (p. 986)
- Entity Reference (p. 989)
- Entity Property Prefixes (p. 1037)
- Creating a New Entity Class (p. 1038)
- Entity Pool System (p. 1039)
- Entity ID Explained (p. 1049)
- Adding Usable Support on an Entity (p. 1049)
- Entity Scripting (p. 1050)
Using the Designer Tool

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 about the latest features, see the Amazon Lumberyard User Guide.

The Designer Tool is an advanced object creation tool. You can easily create complex object meshes with powerful built-in functionality, without the need to use external DCC tools.

Topics
- Designer Tool Settings (p. 968)
- Selection Tools (p. 969)
- Shape Tools (p. 970)
- Edit Tools (p. 972)
- Modify Tools (p. 974)
- Texture Tools (p. 976)
- Miscellaneous Tools (p. 978)
Designer Tool Settings

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 about the latest features, see the Amazon Lumberyard User Guide.

The following parameter groups are available under the Settings panel, on the Objects tab, in the Rollup Bar.

**CD Settings**

The following parameters are available on the CD tab under the Settings panel.

### CD Parameters

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exclusive Mode</td>
<td>Use to make the view look like that of a DCC tool. In this mode, all objects except for the selected objects are hidden and the time of day and light settings are set only. When a level has a lot of objects and is complex, this mode makes the view's complexity decrease.</td>
</tr>
<tr>
<td>Display Back Faces (Editor Only)</td>
<td>Used to enable showing the backfaces of designer objects, such as when the camera is within an object.</td>
</tr>
<tr>
<td>Seamless Edit</td>
<td>Enables editing objects as the mouse cursor hovers over them.</td>
</tr>
<tr>
<td>Keep Pivot Center</td>
<td>Ensures that the pivot remains unaffected during editing.</td>
</tr>
<tr>
<td>Highlight Elements</td>
<td>Toggles visualization of the object's selected elements such as vertices, edges, and faces.</td>
</tr>
<tr>
<td>Highlight Box Size</td>
<td>When Highlight Elements is enabled, this controls the scale of the helpers used to highlight elements.</td>
</tr>
<tr>
<td>Display Dimension Helper</td>
<td>Enables visualization of the object's dimensions, width, height, and depth.</td>
</tr>
<tr>
<td>Display Triangulation</td>
<td>Overlays the object's triangulation.</td>
</tr>
<tr>
<td>Display Subdivided Result</td>
<td>Overlays the object's subdivisions.</td>
</tr>
</tbody>
</table>

**Object Settings**

The following parameters are available on the Object tab under the Settings panel.

### Object Parameters

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cast Shadows</td>
<td>Allows objects to cast shadows</td>
</tr>
<tr>
<td>Support Second Visarea</td>
<td>Normally, objects are considered to be in only one vis area. This option allows them to be added to multiple vis areas if their bounding box overlaps them, at the cost of some performance.</td>
</tr>
</tbody>
</table>
### Property | Description
--- | ---
Without this option, some large objects may not be displayed when viewed through portals in certain situations.
Outdoor | When set, the object will not be rendered when inside a vis area.
Rain Occluder | Occludes dynamic raindrops
View Distance Ratio | Sets the distance from the current view at which the object renders.
AI Exclude From Triangulation | Deprecated
AI Hideable | When this option is set, AI will use this object as a hiding spot, using the specified hide point type.
No Static Decal | When this option is set, decals will not project onto the object.
Exclude Collision | Enable to exclude collisions.
Occluder | Used for the construction of a level occlusion mesh.

## Selection 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 about the latest features, see the Amazon Lumberyard User Guide.

The following function buttons are available from the Selection tab on the Designer Menu panel.

**AllNone**

Use to select or deselect all objects at once.

**Connected**

Use to select all faces connecting one another from the selected face.

**Grow**

Use to expand a selection based on the selected faces. Each time you press Grow, the selection range is enlarged based on the previous selected faces.
Invert

Use to invert the selection states of the faces. Selected faces will be unselected and unselected faces will be selected.

Loop

Use to select serial-linked edges or faces that form a loop from selected edges or faces.

Object

Use to select another object.

Pivot

Use to change the pivot position.

Ring

Use to select sequence edges that are not connected but on the opposite side to each other. You can also select serial-connected quad faces in a direction that is perpendicular to the direction that the selected two faces set.

Vertex, Edge, Face

Use to select and move vertices, edges, and faces. You can select multiple buttons using the Ctrl key.

Shape 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 about the latest features, see the Amazon Lumberyard User Guide.

Note

To learn more about using shapes with the new component entity system, see Shapes in the Amazon Lumberyard User Guide.

The following buttons are available from the SH tab on the Designer Menu panel.
Box

Used to draw one or more boxes. You can adjust the **Width**, **Height**, and **Depth** values.

Cone

Used to draw a cone. You can adjust the **Subdivision Count**, **Height**, and **Radius** values.

Cube Editor

Used to create one or more cubes. You can add, remove, and paint cubes. The following functions are provided:

- **Add** - Add a cube on the brush with the specified Sub Material ID.
- **Remove** - Remove a cube under the brush.
- **Paint** - Paint selected cubes with the specified Sub Material ID.
- **Brush Size** - Select the cube brush size.
- **Sub Material ID** - Specifies the sub material ID. This ID will be recorded to faces affected.
- **Merge Sides** - When enabled, the added faces or remained faces after removing a cube will be merged with the adjoining faces.
Curve

Used to draw either a standard curve or a Bezier curve. You can adjust the Subdivision Count value.

Cylinder

Used to draw a cylinder. You can adjust the Subdivision Count, Height, and Radius values.

Disc

Used to draw a disc. You can adjust the Subdivision Count and Radius values.

Polyline

Used to draw a line or multiple line segments on a surface.

Rectangle

Used to draw a rectangle. You can adjust the Width and Depth values.

Stair

used to create a staircase. You can create stairs having uniform a step size even though the sizes of stairs are different by adjusting a tread size automatically so that a character can rise. The following values can be adjusted:

- **Step Rise** - The size of each step rise.
- **Mirror** - Mirrors a stair against an invisible plane centered.
- **Rotation by 90 Degrees** - Rotates a stair by 90 degrees maintaining the width, height and depth of a box.
- **Width** - The width of the stair.
- **Height** - The height of the stair.
- **Depth** - The depth of the stair.

Stair Profile

Used to draw a stair profile on a surface, which can be pulled using the Extrude function to be a stair. You can adjust the Step Rise value.

Sphere

You can adjust the Subdivision Count and Radius values.

**Edit 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 about the latest features, see the Amazon Lumberyard User Guide.

The following buttons are available from the ED tab on the Designer Menu panel.
**Collapse**

use to collapse all connected edges to the center position.

**Copy**

Use to copy an object face.

**Extrude**

Use to push or pull the selected face so you can expand a 2D surface to a 3D shape.

**Fill**

Use to fill a space based on selected edges or vertices.

**Flip**

use to flip an object face.

**Merge**

Used for merging multiple objects or connected faces to an object or a face.

**Offset**

Used to take a face and create an inset of the selected face.
Modify Tools

Remove

Used to remove selected edges and faces.

Remove Doubles

Used to merge the selected vertices within the specified distance.

Separate

Used to separate two or more objects.

Weld

Used to merge the selected two vertices by moving the first vertex to the second vertex.

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 about the latest features, see the Amazon Lumberyard User Guide.

The following buttons are available from the MO tab on the Designer Menu panel.
Bevel

Used to smooth edges of a shape. Most shapes have blunt edges, so applying the bevel to edges of a shape can add more realism.

Boolean

Select at least two objects, and chose either **Union**, **Difference**, or **Intersection**.

Array Clone

Places cloned objects evenly in a line.

Circle Clone

Places cloned objects in a circle

Lathe

Used to create a mesh by extruding each edge of a profile polygon along a path. You can make a complicated model using this method.
LoopCut

Used for cutting quad-shaped polygons by several loop edges. Set the direction and number of loops. The direction of the loops are set by the edge closest to the cursor and the number of loops are changed by moving the mouse wheel while pressing the CTRL key.

Magnet

Deprecated (merged with the Lathe function).

Mirror

Used to mirror a mesh along an arbitrary plane as well as its local x-, y-, or z-axis plane. This tool has the following functions:

- **Apply** – Splits a mesh by a mirror plane and copies the half part to the other part and then starts the mirror editing.
- **Invert** – Invert a direction of the mirror plane.
- **Center Pivot** – Moves the pivot position to the center of the bounding box.
- **Align X**, **Align Y**, and **Align Z** – Aligns the mirror plane by x-axis, y-axis, or z-axis.
- **Freeze** – Freezes the current geometry.

Subdivision

Used to create a smooth appearance of a mesh without complicated manipulations. A control mesh made this way doesn't need many vertices or faces to model complex smooth surfaces. You can also give each edge a semi-sharp crease, which defines how sharp each edge is.

Texture 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 about the latest features, see the Amazon Lumberyard User Guide.

The following buttons are available from the SU tab on the Designer Menu panel.
Smoothing Group

Used for assigning numbers to faces. Faces with the same numbers and connected by an edge are rendered smoothly. A seam will be displayed between two faces with different smoothing group IDs. The following functions are available:

Parameters

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smoothing Groups</td>
<td>Used to assign a number to the selected faces.</td>
</tr>
<tr>
<td>Add Faces To SG</td>
<td>Used to select faces based on the selected number buttons.</td>
</tr>
<tr>
<td>Select Faces By SG</td>
<td>Used to select faces based on the selected number buttons.</td>
</tr>
<tr>
<td>Clear Empty SGs</td>
<td>Used to remove the assigned smoothing groups of the selected faces.</td>
</tr>
</tbody>
</table>
### Property Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Smooth with Threshold Angle</td>
<td>Sets the smoothing groups based on the angle between faces. Any two faces will be put in the same smoothing group if the angle between their normals is less than the threshold angle.</td>
</tr>
<tr>
<td>Threshold Angle</td>
<td>Used to set the angle in degrees</td>
</tr>
</tbody>
</table>

## UV Mapping

Materials can be assigned to each face differently and you can manipulate the UV coordinates using this tool.

### Parameters

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping</td>
<td></td>
</tr>
<tr>
<td>UV offset</td>
<td>The parameters are set to solid directly.</td>
</tr>
<tr>
<td>Scale offset</td>
<td>The values are added to the existing parameters</td>
</tr>
<tr>
<td>Alignment</td>
<td></td>
</tr>
<tr>
<td>Fit Texture</td>
<td>Fits the texture to the selected surfaces.</td>
</tr>
<tr>
<td>Reset</td>
<td>Resets the texture settings on selected surfaces.</td>
</tr>
<tr>
<td>Tiling</td>
<td>Changes texture tiling on selected surfaces in the X and Y directions.</td>
</tr>
<tr>
<td>Select</td>
<td>Selects all surfaces with the Material ID.</td>
</tr>
<tr>
<td>Assign</td>
<td>Assigns the Material ID to selected surfaces.</td>
</tr>
</tbody>
</table>

## Miscellaneous 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 about the latest features, see the Amazon Lumberyard User Guide.

The following buttons are available from the MI tab on the Designer Menu panel.
Export

Exports .obj, .cgf, or .grp files when these buttons are pressed.

Hide Face

Used to hide or unhide the selected faces.

ResetXForm

Resets the Position, Rotation, or Scale values when these check boxes are selected.

Shortcuts

Used to bind each function/subtool in the Designer tool to specific key combinations. The second column comprises CTRL, SHIFT and CTRL+SHIFT. The last column lists the available virtual keys.
Using the Measurement System Tool

The Measurement System Tool allows to measure the length of segmented objects like roads, rivers, and paths. Measuring of segments is done by following the shape of each segment. The measured path is shown in yellow color.

To read the length of some parts of a segmented object, a start point and an end point must be selected.

To measure a segmented object

1. Click to select the object in the viewport.
2. Click the Edit button. The object should turn yellow and be sunken.
3. Click Tools, Other, Measurement System Tool.
4. Click on the start of your desired first segment and the last segment of your choice to read its length. Double-clicking on any of the segment starting points selects the whole object for measuring or clears the start and end points.
5. Close the tool when done.

Using the Object Selector

Use the Object Selector to select and locate objects such as brushes, entities, tagpoints, volumes, and more. You can also hide and unhide objects, freeze and unfreeze objects, and delete objects. You can perform these actions on objects in layers that are selectable, visible, and not frozen.

To open Object Selector

Do one of the following:

- On the main menu, click Tools, Object Selector.
- Press Ctrl+T.
- In the top toolbar, click the Object Selector icon.
Finding an Object

You may sometimes find it difficult to select an object in your level, particularly when you have a large number of objects, or when other objects are surrounding or overlapping the object you want to select. The **Object Selector** provides several tools to help you find specific objects.

The Object Selector displays objects on layers that are selectable (1), visible (2), and not frozen (3).

To automatically select objects (in your **Perspective** viewport) when you click them in the list, enable the **Auto Select** option (bottom right).

To display objects with parent/child relationships, enable the **Display as Tree** option. When this option is enabled, each type of object is displayed with its icon, and grouped objects are shown as a tree in the list. If you have no grouped objects, you see only individual objects listed.

You can also use **Fast Select** to extend your search to include objects within prefabs and groups. To do this, enable **Search also inside Prefabs and Groups** (below **Fast Select**).

To find and select an object

1. Open (p. 980) the **Object Selector**.
2. Do one or both of the following:
   - If you know the object's name, type it into the **Fast Select** box at the bottom.
   - Select one or more of the **List Types** (on the right):
     - **Entities**
     - **Brushes**
     - **Prefabs**
     - **Tag Points**
3. Click the object(s) you want to select.
4. Click **Select** (on the right) to place an X for each selected object in the **Selected** column.

You can also use:
- **Select All** to select all currently listed objects
- **Select None** to deselect all objects.
- **Invert Selection** to deselect currently selected objects and select all the other listed objects.

5. Close the **Object Selector** to return to your **Perspective** viewport.
6. Press **Z** on your keyboard to focus on the object(s) you selected.

### Managing Objects

The **Object Selector** can also hide (and unhide), freeze (and unfreeze), and delete listed objects. You can perform these actions on objects that are contained in layers that are currently selectable, visible, and not frozen.

#### To hide or freeze objects

1. Find the object(s) you want to hide or freeze.
2. Click the object(s). To select multiple objects, use **Ctrl** or **Shift**.

   **Note**
   For this procedure, you need only click to select. There is no need to click the **Select** button on the right side of the **Object Selector**.

3. Click **Hide** or **Freeze**.

   Clicking **Hide** hides your object(s) in the **Object Selector** list and in your **Perspective** viewport.

   Clicking **Freeze** hides your object(s) in the **Object Selector** list and makes it unable to be interacted with in the **Perspective** viewport.

### Other Actions

<table>
<thead>
<tr>
<th>Task</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>To unhide hidden objects</td>
<td>Select the objects, and then click <strong>Unhide</strong>.</td>
</tr>
<tr>
<td>To view frozen objects</td>
<td>Click the <strong>Frozen</strong> option under <strong>Display List</strong>.</td>
</tr>
<tr>
<td>To unfreeze frozen objects</td>
<td>Select the objects, and then click <strong>Unfreeze</strong>.</td>
</tr>
<tr>
<td>To delete objects</td>
<td>Find the objects, click to select them, and then click <strong>Delete Selected</strong>. This deletes the objects from the <strong>Object Selector</strong> and from your level.</td>
</tr>
</tbody>
</table>
Object Selector Table

The objects in your level are listed in a table in the **Object Selector** window. To sort your displayed objects, click a column header. The results appear in alphabetic order.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the object.</td>
</tr>
<tr>
<td>Selected</td>
<td>X is displayed when object is selected in Perspective viewport.</td>
</tr>
<tr>
<td>Type</td>
<td>Scene element type of the object (entity, brush, prefab, tag point, AI point, group, volume, shape, solid, other).</td>
</tr>
<tr>
<td>Layer</td>
<td>Layer to which the object is assigned (objects on invisible or frozen layers are not displayed).</td>
</tr>
<tr>
<td>Default Material</td>
<td>Path to object's default material.</td>
</tr>
<tr>
<td>Custom Material</td>
<td>Path to object's customer material, if assigned.</td>
</tr>
<tr>
<td>Breakability</td>
<td>Type of breakability the object supports.</td>
</tr>
<tr>
<td>Track View</td>
<td>Track view animation sequence to which the object is used.</td>
</tr>
<tr>
<td>FlowGraph</td>
<td>Flow graph that the object is used in.</td>
</tr>
<tr>
<td>Geometry</td>
<td>Path to the object's geometry, if applicable.</td>
</tr>
<tr>
<td>Instances In Level</td>
<td>Number of times the object is used in the level.</td>
</tr>
<tr>
<td>Number of LODs</td>
<td>Number of <strong>LODs</strong> the object has.</td>
</tr>
<tr>
<td>Spec</td>
<td>The minimum specification that the object is set to display on.</td>
</tr>
<tr>
<td>AI GroupID</td>
<td>Group ID number associated with an AI character.</td>
</tr>
</tbody>
</table>

Brushes

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 about the latest features, see the **Amazon Lumberyard User Guide**.

Brushes are solid objects that cannot be modified or moved dynamically during gameplay, except if they have a break-point specified in the asset file, for example a breakable wooden shack.

Typically brushes are static objects placed in a level. They are one of the cheapest rendered objects as they don't have any of the entity or physics overhead of other objects. A large percentage of the visual objects in your levels will consist of brushes.
# Brush Parameter Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometry</td>
<td>This option specifies the geometry that needs to be used for the object.</td>
</tr>
<tr>
<td><strong>CollisionFiltering</strong></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>• Ship</td>
</tr>
<tr>
<td></td>
<td>• Shield</td>
</tr>
<tr>
<td></td>
<td>• Asteroid</td>
</tr>
<tr>
<td>Ignore</td>
<td>• Ship</td>
</tr>
<tr>
<td></td>
<td>• Shield</td>
</tr>
<tr>
<td></td>
<td>• Asteroid</td>
</tr>
<tr>
<td>OutdoorOnly</td>
<td>When set, the object will not be rendered when inside a vis area.</td>
</tr>
<tr>
<td>CastShadowMaps</td>
<td>When this option is set, the object will cast shadows onto other geometry/terrain/etc.</td>
</tr>
<tr>
<td>RainOccluder</td>
<td>Set the brush to occlude rain, this works in conjunction with Rain Entity. If your level does contain rain, you should set this wisely, as there is a limit of 512 objects that can occlude at any given time.</td>
</tr>
<tr>
<td>SupportSecondVisarea</td>
<td>Normally, objects are considered to be in only one vis area. This option allows them to be added to multiple vis areas if their bounding box overlaps them, at the cost of some performance. Without this option, some large objects may not be displayed when viewed through portals in certain situations.</td>
</tr>
<tr>
<td>Hideable</td>
<td>When this option is set, AI will use this object as a hiding spot, using the specified hide point type.</td>
</tr>
<tr>
<td>LodRatio</td>
<td>Defines how far from the current camera position, the different Level Of Detail models for the object are used.</td>
</tr>
<tr>
<td>ViewDistanceMultiplier</td>
<td>Sets the distance from the current view at which the object renders.</td>
</tr>
<tr>
<td>NotTriangulate</td>
<td>Deprecated</td>
</tr>
<tr>
<td>AirRadius</td>
<td>Deprecated</td>
</tr>
<tr>
<td>NoStaticDecals</td>
<td>When this option is set, decals will not project onto the object.</td>
</tr>
<tr>
<td>NoAmnbShadowCaster</td>
<td>When this option is set, no ambient shadows will be cast.</td>
</tr>
<tr>
<td>RecvWind</td>
<td>When this option is set, the object will be affected by the level wind.</td>
</tr>
<tr>
<td>Occluder</td>
<td>Used for the construction of a level occlusion mesh.</td>
</tr>
<tr>
<td>DrawLast</td>
<td>This function is exposed to give per-object control over alpha-sorting issues. An example can be seen below.</td>
</tr>
</tbody>
</table>
**DrawLast**

The **DrawLast** effects in front of glass objects. By enabling **DrawLast**, Lumberyard knows that any alpha based objects rendered between the player and itself should take ordering priority.

---

**Prefabs**

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 about the latest features, see the Amazon Lumberyard User Guide.

Prefabs are groups of objects that can be placed in the level as instances. An instance is an object that is an exact copy of every other object of the same type. Altering one prefab universally applies the changes to each instance of the prefab object. Any alterations need to be saved to the Prefab Library to ensure they are correctly propagated across the entire game.

The Prefabs Library is a tab in the Database View editor, and lists all the prefab objects that are available for a specific level.

**Prefab Parameter Table**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Global</strong></td>
<td></td>
</tr>
<tr>
<td>Open All</td>
<td>Open all instances of this prefab inside the level.</td>
</tr>
<tr>
<td>Close All</td>
<td>Close all instances of this prefab inside the level.</td>
</tr>
<tr>
<td><strong>Local</strong></td>
<td></td>
</tr>
<tr>
<td>Pick and Attach</td>
<td>Allows you to add a new object to the selected prefab, by clicking on it.</td>
</tr>
<tr>
<td>Delete Object(s)</td>
<td>Allows you to delete one or more objects from the selected prefab.</td>
</tr>
<tr>
<td>Clone Object(s)</td>
<td>Allows you to clone one or more objects from the selected prefab.</td>
</tr>
<tr>
<td>Clone All</td>
<td>Clones all instances of this prefab inside the level.</td>
</tr>
<tr>
<td>Extract Object(s)</td>
<td>Extracts a clone of a single object from the prefab, without altering or removing anything from the prefab object itself.</td>
</tr>
<tr>
<td>Extract All</td>
<td>Extracts all the objects from the prefab, without altering the Prefab Library.</td>
</tr>
<tr>
<td>Open</td>
<td>Opens the prefab group, allowing you to edit and manipulate objects within it.</td>
</tr>
<tr>
<td>Close</td>
<td>Closes the prefab so that internal objects cannot be individually edited.</td>
</tr>
</tbody>
</table>
Common Parameters and Properties

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 about the latest features, see the Amazon Lumberyard User Guide.

Many entities share common parameters and properties, as follows.

**Entity Properties**

Use the Entity pane to modify basic entity properties, such as the name of your object or the currently selected layer. You can type a new name for your object in the first text box in the pane.

Depending on their type, certain entities will have color schemes applied by default. Click the color next to the text box to open the color editor.

Click the layers icon to open the layer window and place your object in the appropriate layer. The layer text box displays the name of the layer that is currently selected.

**Standard Parameters**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area</td>
<td>Increases or decreases the radius of the onscreen object placement helper.</td>
</tr>
<tr>
<td>Mtl button</td>
<td>Opens the material window for you to pick a material to apply to the object. The custom material path displays in the Mtl text box.</td>
</tr>
<tr>
<td>MinSpec</td>
<td>Sets the value at which the selected object appears in game detail settings.</td>
</tr>
</tbody>
</table>

**Entity Parameters**

Use the Entity Params panel to modify common entity parameters. These parameters enable effects that are added to an object, and toggle options such as hiding the object in-game.

**Entity Parameters**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outdoor Only</td>
<td>Prevents an object from rendering when inside a vis area.</td>
</tr>
<tr>
<td>Cast Shadow MinSpec</td>
<td>Casts a shadow on the selected quality setting and above.</td>
</tr>
<tr>
<td>LodRatio</td>
<td>Defines the distance from the current camera position when different level of detail models for the object are used.</td>
</tr>
<tr>
<td>ViewDistRatio</td>
<td>Defines the distance from the current camera position when the object will be rendered.</td>
</tr>
<tr>
<td>HiddenInGame</td>
<td>Prevents an object from displaying in game mode. This is useful for debugging or prototyping.</td>
</tr>
</tbody>
</table>
# Scripting and Flow Graph Entity Parameters

Use this pane to modify the parameters that are related to entity scripting and Flow Graph.

## Scripting and Flow Graph Parameters

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edit Script</td>
<td>Opens the script file in your associated program and allows you to modify the script for the selected entity. Clicking the &gt; button displays more options for this file.</td>
</tr>
<tr>
<td>Reload Script</td>
<td>Implements any changes made to the script. This is useful for reviewing particle effects because reloading reactivates the changes.</td>
</tr>
<tr>
<td>Entity Archetype</td>
<td>Displays the entity name on the button, if the entity is an archetype entity. Opens the archetype in the Database View tool.</td>
</tr>
<tr>
<td>Create</td>
<td>Creates a new flow graph.</td>
</tr>
<tr>
<td>List</td>
<td>Lists the flow graphs that are associated with the selected entity.</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes a flow graph that was created for the entity.</td>
</tr>
<tr>
<td>Sequence</td>
<td>Displays the name of the sequence, if the entity is used in a track view sequence. Opens the sequence in the Track View editor.</td>
</tr>
<tr>
<td>Save State</td>
<td>Saves the physical state of a selected entity (when AI/Physics in enabled). This is useful for placing physical props realistically around your level without manually rotating and aligning their positions.</td>
</tr>
<tr>
<td>Clear State</td>
<td>Clears any saved physical state.</td>
</tr>
</tbody>
</table>

## Entity Links

Use this pane to view the entities that are linked to the main entity. Each entity can link to multiple entities. To create an entity link, you create a dynamic link that can be referenced in Lua script.

To create a link, click **Pick Target** and select the desired entity. You can select multiple entities one at a time while the button is still active.

Double-click a linked entity in the list to select it. Right-click to open a menu with additional commands.
Entity Links Parameters

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Target Entity</td>
<td>Changes the entity associated with a link.</td>
</tr>
<tr>
<td>Rename Link</td>
<td>Renames the selected link.</td>
</tr>
<tr>
<td>Delete Link</td>
<td>Deletes the selected link.</td>
</tr>
<tr>
<td>Pick New Target</td>
<td>Provides the same functionality as the <strong>Pick Target</strong> button.</td>
</tr>
</tbody>
</table>

Entity Events

Use this pane to edit and run the script behind objects. When **AI/Physics** is enabled, you can test the effect of any changes you have made to the entity script.

Enable **AI/Physics** to test events.

Entity Event Parameters

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input/Output</td>
<td>Displays a list of executable script commands.</td>
</tr>
<tr>
<td>Pick New</td>
<td>Deprecated</td>
</tr>
<tr>
<td>Mission Handler</td>
<td>Deprecated</td>
</tr>
<tr>
<td>Remove</td>
<td>Deprecated</td>
</tr>
<tr>
<td>Send</td>
<td>After choosing an input or output event, click <strong>Send</strong> to test the effect. For example, an input event called OnKill might kill an entity and OnSpawn might spawn an entity back to life.</td>
</tr>
<tr>
<td>Methods</td>
<td>Displays a list of executable methods.</td>
</tr>
<tr>
<td>Run</td>
<td>Displays a list of executable methods.</td>
</tr>
<tr>
<td>Goto</td>
<td>Deprecated</td>
</tr>
<tr>
<td>Add</td>
<td>Deprecated</td>
</tr>
</tbody>
</table>

Attached Entities

Use this pane to create links to other objects in the perspective viewport. This pane is visible for certain entities.

Attached Entity Parameters

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pick</td>
<td>Links two selected objects. You will see the link in the viewport and the object name in the target window.</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes a link between two objects.</td>
</tr>
</tbody>
</table>
### Shape Parameters

Use this pane to edit the effect area for a shape and create links to other objects in the viewport. This pane is visible for certain entities.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>Selects an object from the target window. You can also double-click the object name in the target window to select the object.</td>
</tr>
</tbody>
</table>

#### Shape Parameters

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num Points</td>
<td>Relates to the number of points the shape contains in the perspective viewpoint.</td>
</tr>
<tr>
<td>Edit Shape</td>
<td>Allows you to edit the selected shape.</td>
</tr>
<tr>
<td>Use Transform Gizmo</td>
<td>Enables the Transform Gizmo helper.</td>
</tr>
<tr>
<td>Reverse Path</td>
<td>Used with objects like AIPath and reverses the AI path. The arrow onscreen points in the opposite direction to show the new path direction.</td>
</tr>
<tr>
<td>Split</td>
<td>Click two parts of your shape to split your shape and create a new independent shape.</td>
</tr>
<tr>
<td>Reset Height</td>
<td>Flattens the shape and all other points to the height of the selected point.</td>
</tr>
<tr>
<td>Pick</td>
<td>Links a shape to an object. You will see the link in the viewport and the object name in the target window.</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes a link between the selected shape and an object.</td>
</tr>
<tr>
<td>Select</td>
<td>Selects an object from the target window. You can also double-click the object name in the target window to select the object.</td>
</tr>
</tbody>
</table>

### Entity Reference

The following is a complete list of the entities in the Entity system.

#### Topics

- Actor Entity (p. 990)
- AI Control Objects (p. 990)
- Anim Entities (p. 995)
- Archetype Entity (p. 995)
- Area Entities (p. 996)
- Audio Entities (p. 1002)
- Boid Entity (p. 1006)
- Camera Entity (p. 1009)
• Geom Entities (p. 1010)
• Light Entities (p. 1011)
• Lightning Arc Entity (p. 1015)
• Miscellaneous Entities (p. 1017)
• Particle Entities (p. 1019)
• Physics Entities (p. 1020)
• Rain Entity (p. 1029)
• Render Entities (p. 1030)
• River Entity (p. 1030)
• Road Entity (p. 1031)
• Rope Entity (p. 1032)
• Snow Entity (p. 1034)
• Tornado Entity (p. 1035)
• Trigger Entities (p. 1035)

**Actor Entity**

This is a specialized entity that is the basis for characters in a game.

**AI Control Objects**

The following AI entities are provided:

• AI Anchor
• AI Horizontal Occlusion Plane
• AI Path
• AI Perception Modifier
• AI Point
• AI Reinforcement Spot
• AI Shape
• Cover Surface
• Navigation Area
• Navigation Seed Point
• Smart Object
• Tag Point

**AI Anchor**

An AI Anchor is a positional point object that can be used to define specific behaviors for an AI with reference to the location and/or direction of the anchor.

**AI Horizontal Occlusion Plane**

AI agents above and below an AI Horizontal Occlusion Plane will not be able to see through it. It can be used, for example, to restrict an AI on a high ledge from being able to see below the ledge.
## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width</td>
<td>Specifies how wide the entity is.</td>
</tr>
<tr>
<td>Height</td>
<td>Specifies how high the shape area should be (0 means infinite height).</td>
</tr>
<tr>
<td>AreaId</td>
<td>Sets up the ID of the area, so areas with another ID can overlap.</td>
</tr>
<tr>
<td>GroupId</td>
<td>Sets up the Group ID of the area, so areas with another group ID can overlap.</td>
</tr>
<tr>
<td>Priority</td>
<td>Defines the Priority so areas with a higher priority will be processed first.</td>
</tr>
<tr>
<td>Closed</td>
<td>Sets if the area should be closed or if it should be just a line.</td>
</tr>
<tr>
<td>DisplayFilled</td>
<td>Just for visibility in the editor this option defines if the area should be rendered as filled or not.</td>
</tr>
<tr>
<td>DisplaySoundInfo</td>
<td>Enable to expand Sound Obstruction options.</td>
</tr>
<tr>
<td>Agent_height</td>
<td>When Render_voxel_grid is enabled this determines the height along the y-axis of the rendered grid cells.</td>
</tr>
<tr>
<td>Agent_width</td>
<td>When Render_voxel_grid is enabled this determines the height along the x-axis of the rendered grid cells.</td>
</tr>
<tr>
<td>Render_voxel_grid</td>
<td>If true, voxel grid will be rendered when helpers are enabled.</td>
</tr>
<tr>
<td>voxel_offset_x</td>
<td>Offset voxel grid on the x-axis.</td>
</tr>
<tr>
<td>voxel_offset_y</td>
<td>Offset voxel grid on the y-axis.</td>
</tr>
</tbody>
</table>

## AI Path

An AI path is an object which can be used to guide your AI agent along a specific route from point to point in your level.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Road</td>
<td>Defines if the path is to be used by vehicles as a preferred path.</td>
</tr>
<tr>
<td>PathNavType</td>
<td>Sets the AI navigation type of the path. Types of paths available:</td>
</tr>
<tr>
<td></td>
<td>• Flight</td>
</tr>
<tr>
<td></td>
<td>• Free 2D</td>
</tr>
<tr>
<td></td>
<td>• Road</td>
</tr>
<tr>
<td></td>
<td>• Smart Object</td>
</tr>
<tr>
<td></td>
<td>• Triangular</td>
</tr>
<tr>
<td></td>
<td>• Unset</td>
</tr>
<tr>
<td></td>
<td>• Volume</td>
</tr>
<tr>
<td></td>
<td>• Waypoint 3D Surface</td>
</tr>
</tbody>
</table>
### Lumberyard Legacy Reference

#### AI Control Objects

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnchorType</td>
<td>Sets an AI behavior for any AI using the path.</td>
</tr>
<tr>
<td>ValidatePath</td>
<td>Used for 3D Volume paths only, checks and displays path validity in the editor.</td>
</tr>
<tr>
<td>Width</td>
<td>Specifies how wide the entity is.</td>
</tr>
<tr>
<td>Height</td>
<td>Specifies how high the shape area should be (0 means infinite height).</td>
</tr>
<tr>
<td>AreaId</td>
<td>Sets up the ID of the area, so areas with another ID can overlap.</td>
</tr>
<tr>
<td>GroupId</td>
<td>Sets up the Group ID of the area, so areas with another group ID can overlap.</td>
</tr>
<tr>
<td>Priority</td>
<td>Defines the Priority so areas with a higher priority will be processed first.</td>
</tr>
<tr>
<td>Closed</td>
<td>Sets if the area should be closed or if it should be just a line.</td>
</tr>
<tr>
<td>DisplayFilled</td>
<td>Just for visibility in the editor this option defines if the area should be rendered as filled or not.</td>
</tr>
<tr>
<td>DisplaySoundInfo</td>
<td>Enable to expand Sound Obstruction options.</td>
</tr>
<tr>
<td>Agent_height</td>
<td>When Render_voxel_grid is enabled this determines the height along the y-axis of the rendered grid cells.</td>
</tr>
<tr>
<td>Agent_width</td>
<td>When Render_voxel_grid is enabled this determines the height along the x-axis of the rendered grid cells.</td>
</tr>
<tr>
<td>Render_voxel_grid</td>
<td>If true, voxel grid will be rendered when helpers are enabled.</td>
</tr>
<tr>
<td>voxel_offset_x</td>
<td>Offset voxel grid on the x-axis.</td>
</tr>
<tr>
<td>voxel_offset_y</td>
<td>Offset voxel grid on the y-axis.</td>
</tr>
</tbody>
</table>

### AI Perception Modifier

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>Specifies how high the shape area should be (0 means infinite height).</td>
</tr>
<tr>
<td>Closed</td>
<td>Sets if the area should be closed or if it should be just a line.</td>
</tr>
<tr>
<td>DisplayFilled</td>
<td>Just for visibility in the editor this option defines if the area should be rendered as filled or not.</td>
</tr>
</tbody>
</table>

### AI Point

An AI Point is an object that represents a named AI waypoint in your level.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Waypoint, Hide, Sec Hide, Entry/Exit, Exit-only</td>
</tr>
<tr>
<td>Nav Type</td>
<td>Human, 3D Surface</td>
</tr>
<tr>
<td>Removable</td>
<td>Allows AI points to be removed, may be useful for adding entrances for easier traversing.</td>
</tr>
<tr>
<td>Regen Links</td>
<td>Prompts a regeneration of all links in the same navigation region as this one.</td>
</tr>
<tr>
<td>Linked Waypoints</td>
<td>Displays the list of waypoints that are connected to this point.</td>
</tr>
<tr>
<td>Pick</td>
<td>Allows the user to pick a second waypoint to create a permanent AI link.</td>
</tr>
<tr>
<td>Pick impass</td>
<td>Allows the user to pick a second waypoint to create a permanent non-passable link.</td>
</tr>
<tr>
<td>Select</td>
<td>Selects the currently highlighted link in the linked waypoints box.</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes the currently highlighted waypoint links.</td>
</tr>
<tr>
<td>Remove all</td>
<td>Removes all waypoint links from the AI Point.</td>
</tr>
<tr>
<td>Remove all in area</td>
<td>Removes all waypoint links in the nav area.</td>
</tr>
</tbody>
</table>

AI Reinforcement Spot

Defines a point which any relevant AI can use to trigger their reinforcement behavior.

AI Shape

An AI shape is an object which can be used to define an area which AI will use for combat and will search for anchors within.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnchorType</td>
<td>Affects AI behaviors in the same way as the anchors do. The main usage is to check if a point (AI position, target position, etc) is inside a shape of a given AnchorType, in the same way as checking the proximity to an anchor of a given type.</td>
</tr>
<tr>
<td>LightLevel</td>
<td>Affects AI's ability to see (including sight range and speed of detection).</td>
</tr>
<tr>
<td>Width</td>
<td>Specifies how wide the entity is.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Height</td>
<td>Specifies how high the shape area should be (0 means infinite height).</td>
</tr>
<tr>
<td>Areaid</td>
<td>Sets up the ID of the area, so areas with another ID can overlap.</td>
</tr>
<tr>
<td>GroupId</td>
<td>Sets up the Group ID of the area, so areas with another group ID can overlap.</td>
</tr>
<tr>
<td>Priority</td>
<td>Defines the Priority so areas with a higher priority will be processed first.</td>
</tr>
<tr>
<td>Closed</td>
<td>Sets if the area should be closed or if it should be just a line.</td>
</tr>
<tr>
<td>DisplayFilled</td>
<td>Just for visibility in the editor this option defines if the area should be rendered as filled or not.</td>
</tr>
<tr>
<td>DisplaySoundInfo</td>
<td>Enable to expand Sound Obstruction options.</td>
</tr>
<tr>
<td>Agent_height</td>
<td>When Render_voxel_grid is enabled this determines the height along the y-axis of the rendered grid cells.</td>
</tr>
<tr>
<td>Agent_width</td>
<td>When Render_voxel_grid is enabled this determines the height along the x-axis of the rendered grid cells.</td>
</tr>
<tr>
<td>Render_voxel_grid</td>
<td>If true, voxel grid will be rendered when helpers are enabled.</td>
</tr>
<tr>
<td>voxel_offset_x</td>
<td>Offset voxel grid on the x-axis.</td>
</tr>
<tr>
<td>voxel_offset_y</td>
<td>Offset voxel grid on the y-axis.</td>
</tr>
</tbody>
</table>

**Cover Surface**

Cover surfaces can be used to allow the AI agent to take cover in combat situations.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limit Left</td>
<td>The generated cover path to the left side of the cover surface object will be limited to this length.</td>
</tr>
<tr>
<td>Limit Right</td>
<td>The generated cover path to the right side of the cover surface object will be limited to this length.</td>
</tr>
<tr>
<td>Limit Height</td>
<td>The resulting height of all cover surfaces will be limited to this value.</td>
</tr>
</tbody>
</table>

**Navigation Area**

For more information, see the Navigation component.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>The height of the navigation area.</td>
</tr>
</tbody>
</table>
**Smart Object**

An AI Anchor is a point or collection of points which can be used by AI to perform a specific action or event, such as an animation or behavior. Certain smart objects can have special geometry assigned to them, to assist with object placement.

**Tag Point**

An AI Tagpoint is an object used to define a location.

**Anim Entities**

**MannequinObject Entity**

**Entity Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionController</td>
<td>The root object controlling mannequin for a character. It is configured using a controller definition (defining the fragmentIDs, scopes, and scope contexts). It schedules actions onto scopes and holds the global tagstate.</td>
</tr>
</tbody>
</table>

**Archetype Entity**

An Archetype entity is based on a regular entity and specifies individual parameter values for that entity. If the value of an Archetype parameter is changed, all instances of that Archetype in the level are updated automatically.

As such, you can predefine variations of entity classes as Archetype Entities that can be used throughout the game. For global changes affecting all instances, the Archetype Entity just needs to be changed once.

**EntityArchetype Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outdoor Only</td>
<td>When set, the object will not be rendered when inside a vis area.</td>
</tr>
<tr>
<td>Cast Shadow MinSpec</td>
<td>When set, the object will cast a shadow.</td>
</tr>
<tr>
<td>LodRatio</td>
<td>Defines how far from the current camera position, the different Level Of Detail models for the object are used.</td>
</tr>
<tr>
<td>ViewDistanceMultiplier</td>
<td>Defines how far from the current camera position, the object can be seen.</td>
</tr>
<tr>
<td>HiddenInGame</td>
<td>When set, this object is not shown in the pure game mode.</td>
</tr>
<tr>
<td>Receive Wind</td>
<td>When set, this object will be influenced by any wind setup in the level.</td>
</tr>
</tbody>
</table>
Area Entities

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 about the latest features, see the Amazon Lumberyard User Guide.

Area entities are used to create three dimensional zones in the level that can be used to trigger events.

The following area entities can be accessed from the Area button on the Objects tab of the Rollup Bar.

- AreaBox
- AreaSolid
- AreaSphere
- ClipVolume
- OccluderArea
- OccluderPlane
- Portal
- Shape
- VisArea
- WaterVolume

AreaBox

This entity lets you create a box to which you can link triggers and other entities that should be enabled when the player enters or leaves the box.

Parameter Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AreaId</td>
<td>Sets up the ID of the area, so areas with another ID can overlap.</td>
</tr>
<tr>
<td>FadetrnZone</td>
<td>Specifies in meters how big the zone around the box is that is used to fade in the effect attached to the box. Only when the player is inside the box the effect is rendered at 100%, at the beginning of the FadetrnZone its rendered at 0%.</td>
</tr>
<tr>
<td>Width</td>
<td>Specifies how wide the box is.</td>
</tr>
<tr>
<td>Length</td>
<td>Defines how long the box is.</td>
</tr>
<tr>
<td>Height</td>
<td>Specifies how high the shape area should be (0 means infinite height).</td>
</tr>
<tr>
<td>GroupId</td>
<td>Sets up the Group ID of the area, so areas with another group ID can overlap.</td>
</tr>
<tr>
<td>Priority</td>
<td>Defines the Priority so areas with a higher priority will be processed first.</td>
</tr>
<tr>
<td>DisplayFilled</td>
<td>Just for visibility in the editor this option defines if the area should be rendered as filled or not.</td>
</tr>
</tbody>
</table>
## Area Entities

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisplaySoundInfo</td>
<td>Enable to expand Sound Obstruction options.</td>
</tr>
</tbody>
</table>

### AreaSolid

The AreaSolid is for defining complex range of sound obstructions with the Designer tool that is used for geometry editing.

### AreaSphere

The AreaSphere object is used to link triggers and other entities that should be enabled when the player enters or leaves the sphere.

### Parameter Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Areaid</td>
<td>Sets up the ID of the area, so areas with another ID can overlap.</td>
</tr>
<tr>
<td>FadeInZone</td>
<td>Specifies in meters how big the zone around the box is that is used to fade in the effect attached to the box. Only when the player is inside the box the effect is rendered at 100%, at the beginning of the fadeinzone its rendered at 0%.</td>
</tr>
<tr>
<td>Radius</td>
<td>Specifies how big the sphere should be.</td>
</tr>
<tr>
<td>GroupId</td>
<td>Sets up the Group ID of the area, so areas with another group ID can overlap.</td>
</tr>
<tr>
<td>Priority</td>
<td>Defines the Priority so areas with a higher priority will be processed first.</td>
</tr>
<tr>
<td>Filled</td>
<td>Just for visibility in the editor this option defines if the area should be rendered as filled or not.</td>
</tr>
</tbody>
</table>

### Clip Volume

ClipVolumes define geometric shapes that can restrict the influence of lights and cubemaps in a level.

Lights can be associated with ClipVolumes by either placing the light directly inside the object or by creating an entity link from the light to the ClipVolume. Once an association has been established, the **AffectsThisAreaOnly** property on the light source will clip the light's influence to the geometry inside the ClipVolume.

Here are some restrictions on the use of ClipVolume objects:

- The Clip Volume mesh needs to be watertight.
- Clip Volume mesh complexity has an impact on performance.
- ClipVolumes must not overlap.
- Due to performance reasons, forward rendered objects perform the inside test based on their pivot only.
- Each light can be linked to a maximum of two ClipVolumes.
OccluderArea

The OccluderArea object prevents Lumberyard from rendering everything that is behind it. It is used for performance optimization in areas where automatic occlusion from brushes and terrain don’t work very well. This object allows you to create an occlusion plane out of a custom shape with multiple edges, unlike an OccluderPlane object which can only be a square shape.

Parameter Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisplayFilled</td>
<td>Just for visibility in the editor this option defines if the area should be rendered as filled or not.</td>
</tr>
<tr>
<td>CullDistRatio</td>
<td>Specifies at what distance the culling effect should stop occurring.</td>
</tr>
<tr>
<td>UseIndoors</td>
<td>Specifies if the occluder area should be working inside an indoor vis area.</td>
</tr>
</tbody>
</table>

OccluderPlane

The OccluderPlane object is used to occlude objects behind the plane. Like with the OccluderArea object, this typically isn't required because occlusion is done automatically. This object can be used as a fallback method.

Parameter Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>Specifies how high the occluder plane is.</td>
</tr>
<tr>
<td>DisplayFilled</td>
<td>Just for visibility in the editor this option defines if the plane should be rendered as filled or not.</td>
</tr>
<tr>
<td>CullDistRatio</td>
<td>Specifies at what distance the culling effect should stop occurring.</td>
</tr>
<tr>
<td>UseIndoors</td>
<td>Specifies if the occluder plane should work inside a vis area.</td>
</tr>
<tr>
<td>DoubleSide</td>
<td>Specifies if the occluder plane should work from both sides.</td>
</tr>
</tbody>
</table>

Portal

With Portals you can cut holes inside a vis area to create an entrance into a vis area. Portals have to be smaller than the vis area shape but thick enough to protrude both the inside and outside of the vis area, like a door.

You can enable and disable Portals using Flow Graph and you can have multiple Portals in one vis area.

Parameter Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>Specifies how high the portal is.</td>
</tr>
<tr>
<td>DisplayFilled</td>
<td>Just for visibility in the editor this option defines if the area should be rendered as filled or not.</td>
</tr>
</tbody>
</table>
**Parameter** | **Description**
--- | ---
AffectedBySun | Specifies if shadows from the world outside the vis area can travel inside.
IgnoreSkyColor | If this option is turned off the ambient color (sky color in time of day window) is not used indoors.
IgnoreGI | If true, Global Illumination won't be used inside this object.
ViewDistRatio | Specifies how far the vis area is rendered.
SkyOnly | Lets you choose to see only the skybox when you look outside the vis area. If you don't render terrain and outside brushes the performance can be faster so use this option when it is appropriate.
OceanIsVisible | Specifies if the ocean rendering should be visible inside the vis area.
UseDeepness | Specifies if the portal should be working from both sides.
DoubleSide | Specifies if the portal should be working from both sides.

**Shape**

The Shape object lets you create a shape to which you can link triggers and other entities that should be enabled when the player enters or leaves the area shape.

**Parameter Table**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width</td>
<td>Specifies how wide the entity is.</td>
</tr>
<tr>
<td>Height</td>
<td>Specifies how high the shape area should be (0 means infinite height).</td>
</tr>
<tr>
<td>AreaId</td>
<td>Sets up the ID of the area, so areas with another ID can overlap.</td>
</tr>
<tr>
<td>GroupId</td>
<td>Sets up the Group ID of the area, so areas with another group ID can overlap.</td>
</tr>
<tr>
<td>Priority</td>
<td>Defines the Priority so areas with a higher priority will be processed first.</td>
</tr>
<tr>
<td>Closed</td>
<td>Sets if the area should be closed or if it should be just a line.</td>
</tr>
<tr>
<td>DisplayFilled</td>
<td>Just for visibility in the editor this option defines if the area should be rendered as filled or not.</td>
</tr>
<tr>
<td>DisplaySoundInfo</td>
<td>Enable to expand Sound Obstruction options.</td>
</tr>
<tr>
<td>Agent_height</td>
<td>When Render_voxel_grid is enabled this determines the height along the y-axis of the rendered grid cells.</td>
</tr>
<tr>
<td>Agent_width</td>
<td>When Render_voxel_grid is enabled this determines the height along the x-axis of the rendered grid cells.</td>
</tr>
<tr>
<td>Render_voxel_grid</td>
<td>If true, voxel grid will be rendered when helpers are enabled.</td>
</tr>
<tr>
<td>voxel_offset_x</td>
<td>Offset voxel grid on the x-axis.</td>
</tr>
</tbody>
</table>
Area Entities

Parameter Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>voxel_offset_y</td>
<td>Offset voxel grid on the y-axis.</td>
</tr>
</tbody>
</table>

**VisArea**

The vis area object is used to define indoor areas for culling and optimization purposes, as well as lighting. Objects inside a vis area won't be rendered from outside and vice versa, this can help with performance immensely.

Vis areas also can be setup to occlude certain lighting elements such as the sun, which gives flexibility in setting up lighting for your indoor areas.

1. In Rollup Bar, on the **Objects** tab, click **Area, VisArea**.
2. Place the vis area object around the desired area in your level and set the **Height** parameter value. Keep the shape of the vis area as simple as possible.
3. Ensure everything related is inside the vis area.
4. Enable **Snap To Grid**.

**Parameter Table**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>Specifies how high the vis area is.</td>
</tr>
<tr>
<td>DisplayFilled</td>
<td>Just for visibility in the editor this option defines if the area should be rendered as filled or not.</td>
</tr>
<tr>
<td>AffectedBySun</td>
<td>Specifies if shadows from the world outside the vis area can travel inside.</td>
</tr>
<tr>
<td>IgnoreSkyColor</td>
<td>If this option is turned off the ambient color (sky color in time of day window) is not used indoors.</td>
</tr>
<tr>
<td>IgnoreGI</td>
<td>If true, Global Illumination won't be used inside this object.</td>
</tr>
<tr>
<td>ViewDistRatio</td>
<td>Specifies how far the vis area is rendered.</td>
</tr>
<tr>
<td>SkyOnly</td>
<td>Lets you choose to see only the skybox when you look outside the vis area. If you don't render terrain and outside brushes the performance can be faster so use this option when it is appropriate.</td>
</tr>
<tr>
<td>OceanIsVisible</td>
<td>Specifies if the ocean rendering should be visible inside the vis area.</td>
</tr>
</tbody>
</table>

**WaterVolume**

The WaterVolumes object is used for rivers, lakes, pools, puddles, and oceans. For more information about WaterVolumes, see **WaterVolume Shader**.

**Parameter Table**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width</td>
<td>Specifies how wide the entity is.</td>
</tr>
</tbody>
</table>

---

1000
<table>
<thead>
<tr>
<th><strong>Parameter</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>Specifies how high the shape area should be (0 means infinite height).</td>
</tr>
<tr>
<td>Areaid</td>
<td>Sets up the ID of the area, so areas with another ID can overlap.</td>
</tr>
<tr>
<td>GroupId</td>
<td>Sets up the Group ID of the area, so areas with another group ID can overlap.</td>
</tr>
<tr>
<td>Priority</td>
<td>Defines the Priority so areas with a higher priority will be processed first.</td>
</tr>
<tr>
<td>Closed</td>
<td>Sets if the area should be closed or if it should be just a line.</td>
</tr>
<tr>
<td>DisplayFilled</td>
<td>Just for visibility in the editor this option defines if the area should be rendered as filled or not.</td>
</tr>
<tr>
<td>DisplaySoundInfo</td>
<td>Enable to expand Sound Obstruction options.</td>
</tr>
<tr>
<td>Agent_height</td>
<td>When Render_voxel_grid is enabled this determines the height along the y-axis of the rendered grid cells.</td>
</tr>
<tr>
<td>Agent_width</td>
<td>When Render_voxel_grid is enabled this determines the height along the x-axis of the rendered grid cells.</td>
</tr>
<tr>
<td>Render_voxel_grid</td>
<td>If true, voxel grid will be rendered when helpers are enabled.</td>
</tr>
<tr>
<td>voxel_offset_x</td>
<td>Offset voxel grid on the x-axis.</td>
</tr>
<tr>
<td>voxel_offset_y</td>
<td>Offset voxel grid on the y-axis.</td>
</tr>
<tr>
<td>Depth</td>
<td>Sets the depth of the river.</td>
</tr>
<tr>
<td>Speed</td>
<td>Defines how fast physicalized objects move along the river. Use negative values to move in the opposite direction.</td>
</tr>
<tr>
<td>UScale</td>
<td>Sets the texture tiling on the U axis.</td>
</tr>
<tr>
<td>VScale</td>
<td>Sets the texture tiling on the V axis.</td>
</tr>
<tr>
<td>View Distance Multiplier</td>
<td>Sets the distance from the current view at which the object renders.</td>
</tr>
<tr>
<td>Caustics</td>
<td>Enables optical caustics effects.</td>
</tr>
<tr>
<td>CausticIntensity</td>
<td>Scales the intensity of the caustics for the water surface normals.</td>
</tr>
<tr>
<td>CausticTiling</td>
<td>Scales the caustic tiling applied to the water surface normals. It allows the scaling of caustics independently from the surface material.</td>
</tr>
<tr>
<td>CausticHeight</td>
<td>Sets the height above the water surface at which caustics become visible. Use this to make caustics appear on overhanging landforms or vegetation and other nearby objects.</td>
</tr>
<tr>
<td><strong>Advanced</strong></td>
<td></td>
</tr>
<tr>
<td>FixedVolume</td>
<td>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).</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>VolumeAccuracy</td>
<td>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).</td>
</tr>
<tr>
<td>ExtrudeBorder</td>
<td>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.</td>
</tr>
<tr>
<td>ConvexBorder</td>
<td>Takes convex hull of the border. This is useful if the border would otherwise have multiple contours, which areas do not support.</td>
</tr>
<tr>
<td>ObjectSizeLimit</td>
<td>Only objects with a volume larger than this number takes part in water displacement (set in fractions of FixedVolume).</td>
</tr>
<tr>
<td>WaveSimCell</td>
<td>Size of cell for wave simulation (0 means no waves). Can be enabled regardless of whether FixedVolume is used.</td>
</tr>
<tr>
<td>WaveSpeed</td>
<td>Sets how &quot;fast&quot; the water appears.</td>
</tr>
<tr>
<td>WaveDamping</td>
<td>Standard damping.</td>
</tr>
<tr>
<td>WaveTimestep</td>
<td>This setting may need to be decreased to maintain stability if more aggressive values for speed are used.</td>
</tr>
<tr>
<td>MinWaveVel</td>
<td>Sleep threshold for the simulation.</td>
</tr>
<tr>
<td>DepthCells</td>
<td>Sets the depth of the moving layer of water (in WaveSimCell units). Larger values make waves more dramatic.</td>
</tr>
<tr>
<td>HeightLimit</td>
<td>Sets a hard limit on wave height (in WaveSimCell units).</td>
</tr>
<tr>
<td>Resistance</td>
<td>Sets how strongly moving objects transfer velocity to the water.</td>
</tr>
<tr>
<td>SimAreaGrowth</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Audio Entities**

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 about the latest features, see the Amazon Lumberyard User Guide.

This topic describes how to use legacy audio entities in the Rollup Bar. These audio entities will be deprecated and replaced with audio component entities in a future release. For more information, see the Component Reference.

There are four Audio entities, as follows:

- Audio Trigger Spot Entity
Audio Entities

- Audio Area Entity
- Audio Area Ambience Entity
- Audio Area Random Entity

Audio Trigger Spot

This topic describes how to use the legacy audio trigger spot in the Rollup Bar. This audio entity will be deprecated and replaced with audio trigger component entity in a future release. For more information, see the Audio Trigger component.

The AudioTriggerSpot triggers an event on a specific position. This position can be automatically randomized on each axis or with time delays.

Audio Trigger Spot Properties

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Defines whether the Entity is enabled (playing) or disabled (not playing).</td>
</tr>
<tr>
<td>MaxDelay</td>
<td>The maximum delay in seconds that it takes to trigger the sound when PlayRandom is enabled.</td>
</tr>
<tr>
<td>MinDelay</td>
<td>The minimum delay in seconds that it takes to trigger the sound when PlayRandom is enabled.</td>
</tr>
<tr>
<td>PlayOnX</td>
<td>Defines whether the sound gets positioned randomly on the x-axis when PlayRandom is enabled.</td>
</tr>
<tr>
<td>PlayOnY</td>
<td>Defines whether the sound gets positioned randomly on the y-axis when PlayRandom is enabled.</td>
</tr>
<tr>
<td>PlayOnZ</td>
<td>Defines whether the sound gets positioned randomly on the z-axis when PlayRandom is enabled.</td>
</tr>
<tr>
<td>PlayRandom</td>
<td>When the check box is enabled: The sound is triggered at random intervals between the MinDelay and MaxDelay settings used and on the PlayOnX, PlayOnY, or PlayOnZ axis that has been selected. When the check box is not enabled, the sound is played immediately on the entity.</td>
</tr>
<tr>
<td>PlayTriggerName</td>
<td>Name of the play event.</td>
</tr>
<tr>
<td>RadiusRandom</td>
<td>The radius in meters in which the sound gets positioned randomly when PlayRandom is enabled.</td>
</tr>
<tr>
<td>SerializePlayState</td>
<td>Defines whether the play state of the entity gets saved and loaded at checkpoints.</td>
</tr>
<tr>
<td>SoundObstructionType</td>
<td>Sets the number of ray casts that are used to calculate the obstruction. More ray casts used equals a greater performance requirement, but creates a more accurate result.</td>
</tr>
</tbody>
</table>

- Ignore – No raycasts are applied and the sound is unaffected by other objects in the game.
Audio Entities

Properties

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Defines whether the Entity is enabled (playing) or disabled (not playing).</td>
</tr>
<tr>
<td>Environment</td>
<td>Defines the name of the ATL environment used inside the connected shape.</td>
</tr>
<tr>
<td>EnvironmentDistance</td>
<td>The distance in meters from the edge of the assigned shape where the fading of the Environment begins.</td>
</tr>
<tr>
<td>FadeDistance</td>
<td>The distance in meters from the edge of the assigned shape where the flow graph node is starting to output values.</td>
</tr>
<tr>
<td>SoundObstructionType</td>
<td>Sets the number of ray casts that are used to calculate the obstruction. More ray casts used equals a greater performance requirement, but creates a more accurate result.</td>
</tr>
</tbody>
</table>

- **Ignore** – No raycasts are applied and the sound is unaffected by other objects in the game.
- **Single Ray** – Used to calculate the treatment the sound receives depending on the position and physical properties of the objects found between the source and the listener.
- **Multiple Rays** – Used to calculate the treatment the sound receives depending on the position and physical properties of the objects found between the source and the listener.

StopTriggerName | Name of the stop event.

Audio Area Entity

This topic describes how to use the legacy audio area entity in the Rollup Bar. This audio entity will be deprecated and replaced with audio area environment component entity in a future release. For more information, see the Audio Area Environment component.

Audio Area Entities are used to play ambient sounds in an area, and are linked to Area Shapes, Area Boxes, and Area Spheres.

These entities are an advanced method of setting up ambient sounds in levels and require Flow Graph logic to play and control the sounds. This opens up many possibilities and gives advanced control over the ambience. When setting up a basic ambient sound, use the Audio Area Ambience entity instead, which does not require any Flow Graph logic.

Audio Area Entity Properties

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Defines whether the Entity is enabled (playing) or disabled (not playing).</td>
</tr>
<tr>
<td>Environment</td>
<td>Defines the name of the ATL environment used inside the connected shape.</td>
</tr>
<tr>
<td>EnvironmentDistance</td>
<td>The distance in meters from the edge of the assigned shape where the fading of the Environment begins.</td>
</tr>
<tr>
<td>FadeDistance</td>
<td>The distance in meters from the edge of the assigned shape where the flow graph node is starting to output values.</td>
</tr>
<tr>
<td>SoundObstructionType</td>
<td>Sets the number of ray casts that are used to calculate the obstruction. More ray casts used equals a greater performance requirement, but creates a more accurate result.</td>
</tr>
</tbody>
</table>

- **Ignore** – No raycasts are applied and the sound is unaffected by other objects in the game.
- **Single Ray** – Used to calculate the treatment the sound receives depending on the position and physical properties of the objects found between the source and the listener.
- **Multiple Rays** – Used to calculate the treatment the sound receives depending on the position and physical properties of the objects found between the source and the listener.
Audio Area Ambience

This topic describes how to use the legacy audio area ambience entity in the Rollup Bar. This audio entity will be deprecated and replaced with audio environment component entity in a future release. For more information, see the Audio Environment component.

Audio Area Ambience entities are used to set up ambiences without having to define their functionality in Flow Graph. They are used when setting up basic ambient shapes in levels that do not require a more complex functionality.

**Audio Area Ambience Properties**

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Defines whether the Entity is enabled (playing) or disabled (not playing).</td>
</tr>
<tr>
<td>Environment</td>
<td>Defines the name of the ATL environment used inside the connected shape.</td>
</tr>
<tr>
<td>EnvironmentDistance</td>
<td>The distance in meters from the edge of the assigned shape where the fading of the environment begins.</td>
</tr>
<tr>
<td>PlayTrigger</td>
<td>Name of the play event.</td>
</tr>
<tr>
<td>Rtpc</td>
<td>Sets the RTPC that is controlling the playing of the sound object.</td>
</tr>
<tr>
<td>RtpcDistance</td>
<td>The distance in meters from the edge of the assigned shape where the connected RTPC is starting to receive values. The values sent to the RTPC are always from 0 to 1.</td>
</tr>
<tr>
<td>SoundObstructionType</td>
<td>Sets the number of ray casts that are used to calculate the obstruction. More ray casts used equals a greater performance requirement, but creates a more accurate result.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Ignore</strong> – No raycasts are applied and the sound is unaffected by other objects in the game.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Single Ray</strong> – Used to calculate the treatment the sound receives depending on the position and physical properties of the objects found between the source and the listener.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Multiple Rays</strong> – Used to calculate the treatment the sound receives depending on the position and physical properties of the objects found between the source and the listener.</td>
</tr>
<tr>
<td>StopTrigger</td>
<td>Name of the stop event.</td>
</tr>
</tbody>
</table>

Audio Area Random

This topic describes how to use the legacy audio area random entity in the Rollup Bar. This audio entity will be deprecated and replaced with audio area environment component entity in a future release. For more information, see the Audio Environment component.
Audio Area Random entities trigger randomized shots in a confined area. The Entity needs to be linked to Area Shapes, Area Boxes, or Area Spheres. The sound is randomly triggered and positioned in a radius around the listener, providing they are inside the connected area.

**Audio Area Random Properties**

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Defines whether the entity is enabled (playing) or disabled (not playing).</td>
</tr>
<tr>
<td>MaxDelay</td>
<td>The maximum delay in seconds it takes to trigger the sound.</td>
</tr>
<tr>
<td>MinDelay</td>
<td>The minimum delay in seconds it takes to trigger the sound.</td>
</tr>
<tr>
<td>MoveWithEntity</td>
<td>When enabled, the sound moves in relation to the listener after it has spawned; otherwise, it stays at its initial position.</td>
</tr>
<tr>
<td>PlayTrigger</td>
<td>Name of the play event.</td>
</tr>
<tr>
<td>RadiusRandom</td>
<td>Defines the size of the radius in which sounds spawn around the listener.</td>
</tr>
<tr>
<td>Rtpc</td>
<td>Sets the RTPC that is controlling the playing of the sound.</td>
</tr>
<tr>
<td>RtpcDistance</td>
<td>The distance in meters from the edge of the assigned shape where the connected RTPC is starting to receive values. The values sent to the RTPC range from 0 to 1.</td>
</tr>
<tr>
<td>SoundObstructionType</td>
<td>Sets the number of ray casts that are used to calculate the obstruction. More ray casts used equals a greater performance requirement, but creates a more accurate result.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Ignore</strong> – No raycasts are applied and the sound is unaffected by other objects in the game.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Single Ray</strong> – Used to calculate the treatment the sound receives depending on the position and physical properties of the objects found between the source and the listener.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Multiple Rays</strong> – Used to calculate the treatment the sound receives depending on the position and physical properties of the objects found between the source and the listener.</td>
</tr>
<tr>
<td>StopTrigger</td>
<td>Name of the stop event.</td>
</tr>
</tbody>
</table>

**Note**

For each audio object, it is a good practice to see which **SoundObstructionType** works best and to select **Ignore** when there is no advantage gained from having obstruction and occlusion values calculated. Select **MultipleRays** only if the accuracy of the single raycast is not sufficient, or if you want the entity to be able to calculate both the occlusion and obstruction values separately.

Raycasts are skipped for entities that do not have an active playing trigger, even when the **SoundObstructionType** is set to **SingleRay** or **MultipleRays**.

**Boid Entity**

Boid entities simulate animals exhibiting group behavior, obstacle avoidance, animations, and sound. Their complex behavior arises from the interaction of an individual agent boid with other boids and the environment in which they move.
Not all parameters are available for all boid classes. For example, Behavior classes are needed only for the Bugs boid class and do not appear in other Boid properties.

### Boid Properties

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model1-5</td>
<td>Additional geometry for the boid; this can be a character (.CHR) or static geometry (.CGF). If you specify more than one option, the geometry is selected at random.</td>
</tr>
<tr>
<td>Model</td>
<td>Geometry for the boid; this can be a character (.CHR) or static geometry (.CGF).</td>
</tr>
<tr>
<td>Mass</td>
<td>Mass of each individual boid.</td>
</tr>
<tr>
<td>Invulnerable</td>
<td>Specifies whether the boid can be killed or not.</td>
</tr>
<tr>
<td>gravity_at_death</td>
<td>Gravity acceleration that affects the body of the killed boid.</td>
</tr>
<tr>
<td>Count</td>
<td>Specifies how many individual objects are spawned.</td>
</tr>
<tr>
<td>Behavior</td>
<td>Movement behavior for the boid entity:</td>
</tr>
<tr>
<td></td>
<td>• 0 = Generic ground bugs, such as beetles</td>
</tr>
<tr>
<td></td>
<td>• 1 = Flying insects, such as dragonflies</td>
</tr>
<tr>
<td></td>
<td>• 2 = Leaping insects, such as grasshoppers</td>
</tr>
</tbody>
</table>

### Flocking Properties

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttractDistMax</td>
<td>Maximum distance at which one boid can see another boid. Boids that are too far away are not interacted with.</td>
</tr>
<tr>
<td>AttractDistMin</td>
<td>Minimum distance that boids are comfortable with to stay close to each other before the separation force starts to affect them.</td>
</tr>
<tr>
<td>EnableFlocking</td>
<td>When enabled, the rules of the emergent flocking behavior is calculated on the whole flock of boids.</td>
</tr>
<tr>
<td>FactorAlign</td>
<td>Steer towards the average heading of local flock-mates.</td>
</tr>
<tr>
<td>FactorCohesion</td>
<td>Steer to move toward the average position of local flock-mates.</td>
</tr>
<tr>
<td>FactorSeparation</td>
<td>Steer to avoid crowding local flock-mates, only when closer then AttractDistMin.</td>
</tr>
<tr>
<td>FieldOfViewAngle</td>
<td>Field of vision of the boid to consider other boids as flock-mates.</td>
</tr>
</tbody>
</table>

**Note**
The following Ground properties apply only when boids are walking on the ground. Boids are able to land only in game mode and not while editing.
## Ground Properties

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WalkToIdleDuration</td>
<td>Time it takes for boids to transition from walking to idle state.</td>
</tr>
<tr>
<td>WalkSpeed</td>
<td>Walk speed when boids land.</td>
</tr>
<tr>
<td>OnGroundWalkDurationMin</td>
<td>Minimum time that boids can spend in walk state.</td>
</tr>
<tr>
<td>OnGroundWalkDurationMax</td>
<td>Maximum time that boids can spend in walk state.</td>
</tr>
<tr>
<td>OnGroundIdleDurationMin</td>
<td>Minimum time that boids can spend in idle state.</td>
</tr>
<tr>
<td>OnGroundIdleDurationMax</td>
<td>Maximum time that boids can spend in idle state.</td>
</tr>
<tr>
<td>HeightOffset</td>
<td>Vertical offset of boids from the ground.</td>
</tr>
<tr>
<td>FactorSeparation</td>
<td>Tries to ensure that boids avoid one another.</td>
</tr>
<tr>
<td>FactorOrigin</td>
<td>Controls how much boids are attracted to their point of origin.</td>
</tr>
<tr>
<td>FactorCohesion</td>
<td>Tries to ensure that boids group together.</td>
</tr>
<tr>
<td>FactorAlign</td>
<td>Tries to ensure that all boids move in roughly the same direction.</td>
</tr>
</tbody>
</table>

## Movement Properties

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FactorAvoidLand</td>
<td>Force coefficient to divert boid from the land or water.</td>
</tr>
<tr>
<td>FactorHeight</td>
<td>Controls the force that is applied to keep boids at the original height for the flock.</td>
</tr>
<tr>
<td>FactorOrigin</td>
<td>Controls the force that attract boids to the origin point of the flock.</td>
</tr>
<tr>
<td>FactorTakeOff</td>
<td>Vertical movement speed scale during take-off.</td>
</tr>
<tr>
<td>FlightTime</td>
<td>Approximate flight time before attempting to land.</td>
</tr>
<tr>
<td>HeightMax</td>
<td>Maximal height boids can fly to (height above land).</td>
</tr>
<tr>
<td>HeightMin</td>
<td>Minimal height boid can fly at (height above land).</td>
</tr>
<tr>
<td>LandDecelerationHeight</td>
<td>Height at which boids start to decelerate when landing.</td>
</tr>
<tr>
<td>MaxAnimSpeed</td>
<td>If the boid had animations, then use this variable to control the speed of the animation.</td>
</tr>
<tr>
<td>SpeedMax</td>
<td>Maximum speed for boid movement.</td>
</tr>
<tr>
<td>SpeedMin</td>
<td>Minimum speed for boid movement.</td>
</tr>
</tbody>
</table>
Options Properties

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate</td>
<td>When checked, active boids are visible and move from the start of the level; alternatively, boids can be activated at a later stage with the activate event.</td>
</tr>
<tr>
<td>AnimationDist</td>
<td>Maximum distance from camera at which animations update.</td>
</tr>
<tr>
<td>FollowPlayer</td>
<td>When checked, boids wrap around only current player position, and the flock origin point becomes the player position. If the boid flies too far away from the player, it reappears on the opposite side.</td>
</tr>
<tr>
<td>NoLanding</td>
<td>Turns landing for birds flocks on and off.</td>
</tr>
<tr>
<td>ObstacleAvoidance</td>
<td>Boids sense the physical environment and can be diverted from the physical obstacles. This option adds heavier physical checks on the boids and should be used carefully (only when really needed).</td>
</tr>
<tr>
<td>Radius</td>
<td>Maximum radius that the boid can move from the flock origin point.</td>
</tr>
<tr>
<td>SpawnFromPoint</td>
<td>If true, all the boids spawn at the boid entity position.</td>
</tr>
<tr>
<td>StartOnGround</td>
<td>If true, boids spawn on the ground; otherwise, they spawn in the air.</td>
</tr>
<tr>
<td>VisibilityDist</td>
<td>Maximum distance from which the whole flock can be visible. If player camera is further away from the flock origin point than VisibilityDist, boids are not simulated and rendered.</td>
</tr>
</tbody>
</table>

ParticleEffect Properties

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EffectScale</td>
<td>Scale of the particle effect to be played.</td>
</tr>
<tr>
<td>waterJumpSplash</td>
<td>Particle effect to be played when the boid splashes into the water.</td>
</tr>
</tbody>
</table>

Camera 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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the (legacy) Camera entity to place a camera in your level. In addition to placing a camera, you can also place a 'LookAt' target, which automatically determines where the camera faces. For more information about this legacy camera entity, see Rendering Cameras.

To use the newer Camera components, see the Camera component and the Camera Rig component.

To add a render camera to your level

1. In the Rollup Bar, on the Objects tab, click Misc, Camera.
2. Hover your pointer in your level, and then click to position the camera.
3. To create a 'LookAt' target (determines what the camera looks at), when placing the camera in your level, hover your pointer to determine its position, then click and drag to the 'LookAt' target, or the view that you want your camera to see. A 'LookAt' target is created where the drag ended. This object is named with your original camera name, with Target appended.

4. Adjust the values of the following parameters as needed.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOV</td>
<td>The vertical field of view of the camera</td>
</tr>
<tr>
<td>NearZ</td>
<td>The cut off point closest to the camera</td>
</tr>
<tr>
<td>FarZ</td>
<td>The max cut off point of the camera</td>
</tr>
</tbody>
</table>

**Shake Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amplitude A</td>
<td>Strength of the effect on each axis</td>
</tr>
<tr>
<td>Amplitude A Multiplier</td>
<td>Multiplier for the amplitude.</td>
</tr>
<tr>
<td>Frequency A</td>
<td>How often the effect plays on each axis</td>
</tr>
<tr>
<td>Frequency A Multiplier</td>
<td>Multiplier for the frequency</td>
</tr>
<tr>
<td>Noise A Amplitude Multiplier</td>
<td>Adds some noise to the amplitude value</td>
</tr>
<tr>
<td>Noise A Frequency Multiplier</td>
<td>Adds some noise to the frequency value</td>
</tr>
<tr>
<td>Time Offset A</td>
<td>A time offset</td>
</tr>
<tr>
<td>Amplitude B</td>
<td>Strength of the effect on each axis</td>
</tr>
<tr>
<td>Amplitude B Multiplier</td>
<td>Multiplier for the amplitude.</td>
</tr>
<tr>
<td>Frequency B</td>
<td>How often the effect plays on each axis</td>
</tr>
<tr>
<td>Frequency B Multiplier</td>
<td>Multiplier for the frequency</td>
</tr>
<tr>
<td>Noise B Amplitude Multiplier</td>
<td>Adds some noise to the amplitude value</td>
</tr>
<tr>
<td>Noise B Frequency Multiplier</td>
<td>Adds some noise to the frequency value</td>
</tr>
<tr>
<td>Time Offset B</td>
<td>A time offset</td>
</tr>
<tr>
<td>Random Seed</td>
<td>Apply some random variation to the noise</td>
</tr>
</tbody>
</table>

**Geom Entities**

A Geom Entity is a very simple entity that takes its physical parameters from its assigned geometry. They are interactive entities with physical values, so they behave like real life objects. It is similar to a Basic Entity, but simpler, more efficient, and has fewer configurable parameters. Geom Entities that have physical properties set in the asset will get pushed away or break up in explosions, for example.

Navigate through the object library browser and drag the desired object to your level.
Light Entities

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 about the latest features, see the Amazon Lumberyard User Guide.

Light Entity

Use the Light entity to control the lights in your level.

Entity Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Turns the light on or off.</td>
</tr>
<tr>
<td>AttenuationBulbSize</td>
<td>If you use the AmbientLight property, set the AttenuationBulbSize value to 0 to revert to the non-physical attenuation model. For more information, see Attenuation and Falloff.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Color</strong></td>
<td></td>
</tr>
<tr>
<td>Diffuse</td>
<td>Specifies the RGB diffuse color of the light.</td>
</tr>
<tr>
<td>DiffuseMultiplier</td>
<td>Controls the strength of the diffuse color.</td>
</tr>
<tr>
<td>SpecularMultiplier</td>
<td>Controls the strength of the specular brightness.</td>
</tr>
<tr>
<td><strong>Options</strong></td>
<td></td>
</tr>
<tr>
<td>AffectThisAreaOnly</td>
<td>If false, casts the light in multiple visual areas.</td>
</tr>
<tr>
<td>AffectVolumetricFogOnly</td>
<td>Enables the light to affect only volumetric fog and not meshes.</td>
</tr>
<tr>
<td>AmbientLight</td>
<td>Enables the light behave like an ambient light source, with no point of origin.</td>
</tr>
<tr>
<td>FakeLight</td>
<td>Disables light projection. This is useful for lights that have only flare effects.</td>
</tr>
<tr>
<td>FogRadialLobe</td>
<td>Adjusts the blend ratio of the main radial lobe (parallel to the eye ray) and side radial lobe (perpendicular to the eye ray). The direction of the main radial lobe depends on the <strong>Anisotropic</strong> parameter value that is used in the Time of Day Editor.</td>
</tr>
<tr>
<td>ForceDisableCheapLight</td>
<td>Declassifies the light as a CheapLight, which is memory optimization during export for the Pure Game mode. Lights are automatically declassified based on the tool that uses them, so this property is optional.</td>
</tr>
<tr>
<td>IgnoresVisAreas</td>
<td>Determines if the light responds to visual areas.</td>
</tr>
<tr>
<td>VolumetricFog</td>
<td>Enables the light to affect volumetric fog.</td>
</tr>
<tr>
<td><strong>Projector</strong></td>
<td></td>
</tr>
<tr>
<td>ProjectorFov</td>
<td>Specifies the angle at which the light texture is projected.</td>
</tr>
<tr>
<td>ProjectorNearPlane</td>
<td>Sets the near plane for the projector. Surfaces that are closer to the light source than the specified value will not be projected on.</td>
</tr>
<tr>
<td>Texture</td>
<td>Sets the texture to project in the direction of the y-axis for the light entity. The light projector texture must have a resolution of 512 x 512 pixels and may not contain an alpha channel.</td>
</tr>
<tr>
<td><strong>Shadows</strong></td>
<td></td>
</tr>
<tr>
<td>CastShadows</td>
<td>Casts shadows based on the configuration specification. Set the value to Low Spec to always cast shadows. This setting does not determine the quality setting for shadows. If you use tiled shading, the amount of shadow-casting lights is limited to 12 by default. Each set of four lights requires an additional 8 MB of memory for shadow texture mapping. You can use the r_ShadowCastingLightsMaxCount console variable to modify the limit value.</td>
</tr>
<tr>
<td>ShadowBias</td>
<td>Moves the shadow cascade toward or away from the shadow-casting object.</td>
</tr>
</tbody>
</table>
### Property

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShadowMinResPercent</td>
<td>Specifies, per light, the percentage of the shadow pool that the light uses for shadows. Use the default value for better performance over quality.</td>
</tr>
<tr>
<td>ShadowSlopeBias</td>
<td>Adjusts the gradient (slope-based) bias that is used to compute the shadow bias.</td>
</tr>
<tr>
<td>ShadowUpdateMinRadius</td>
<td>Defines the minimum radius of the light source to the player camera where the ShadowUpdateRatio setting is ignored. For example, if the minimum radius value is set to 10 and the camera is less than 10 meters from the light source, the shadow will update normally. If the camera is farther than 10 meters from the light source, the shadow will use the ShadowUpdateRatio value. This property does not work with the Very High specification because shadow caching is disabled.</td>
</tr>
<tr>
<td>ShadowUpdateRatio</td>
<td>Defines the update ratio for shadow maps that are cast from the light. Lower values produce less frequent updates and more stuttering. This property is enabled or disabled based on the value for ShadowUpdateMinRadius, and the distance of the player camera from the light source. This property does not work with the Very High specification because shadow caching is disabled.</td>
</tr>
</tbody>
</table>

### Shape

<table>
<thead>
<tr>
<th>Shape</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PlanarLight</td>
<td>Makes the selected light entity become an area light. You can set the r_DeferredShadingAreaLights console variable to 1 to use area or planar lights.</td>
</tr>
</tbody>
</table>

### Style

<table>
<thead>
<tr>
<th>Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnimationPhase</td>
<td>Starts the light animation that has a specific light style property, at a different point along the sequence. This is useful when you have multiple lights that use the same animation in the same scene. This property allows the animations to play asynchronously.</td>
</tr>
<tr>
<td>AnimationSpeed</td>
<td>Specifies the speed at which the light animation should play.</td>
</tr>
<tr>
<td>AttachToSun</td>
<td>Allows the sun to use the flare properties for the light.</td>
</tr>
<tr>
<td>Flare</td>
<td>Specifies the path to the item in the flare library.</td>
</tr>
<tr>
<td>FlareEnable</td>
<td>Enables the Flare Editor to use the flare properties.</td>
</tr>
<tr>
<td>FlareFOV</td>
<td>Controls the field of view (FOV) for the flare. You must enable this property for the individual flare.</td>
</tr>
<tr>
<td>LightAnimation</td>
<td>Specifies the track view sequence to use to animate the light.</td>
</tr>
<tr>
<td>LightStyle</td>
<td>Specifies the preset animation for the light to play. You can use the Light.cfx shader to define the styles. Valid values: 0 to 48. The testing and debug styles are values 40 to 48.</td>
</tr>
</tbody>
</table>

### Environment Probe Entity

You can use environment probes (also known as light probes) to place cubemaps throughout a level. This is useful to automatically assign cubemaps to any element within the radius.
## Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Enables or disables the probe.</td>
</tr>
<tr>
<td>BoxSizeX, BoxSizeY, BoxSizeZ</td>
<td>Specifies the xyz dimensions of the probe's area of effect. Probes are projected as cubes in the level. For a global probe, set the values high enough to span the entire level.</td>
</tr>
<tr>
<td>Diffuse</td>
<td>Sets the diffuse color of the light. Set the value to 255, 255, 255.</td>
</tr>
<tr>
<td>DiffuseMultiplier</td>
<td>Increases the brightness of the light. Set the value to 1.</td>
</tr>
<tr>
<td>SpecularMultiplier</td>
<td>Multiplies the specular color brightness. Set the value to 1.</td>
</tr>
<tr>
<td>AffectsThisAreaOnly</td>
<td>Determines the affected area. Set the value to False for the lights to cover other visual areas.</td>
</tr>
<tr>
<td>AttenuationFalloffMax</td>
<td>Determines the falloff amount to create smoother transitions or hard edges. Valid values: 0 to 1. A value of 0.8 indicates that falloff begins at 80% at the boundaries of the box. Set the value to 0 for a global probe (no falloff).</td>
</tr>
<tr>
<td>IgnoresVisAreas</td>
<td>Determines whether the light responds to visual areas. Set the value to True for a global probe.</td>
</tr>
<tr>
<td>SortPriority</td>
<td>Determines which probe has more visual interest and a higher priority. Set the value to 0 for a global probe and then increase the value for local probes. Higher values indicate more localized probes.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deferred_cubemap</td>
<td>Specifies the file location of the cubemap texture.</td>
</tr>
<tr>
<td>BoxHeight</td>
<td>Adjusts the height of the cubemap box.</td>
</tr>
<tr>
<td>BoxLength</td>
<td>Adjusts the length of the cubemap box.</td>
</tr>
<tr>
<td>BoxProject</td>
<td>Considers the size of the cubemap box.</td>
</tr>
<tr>
<td>BoxWidth</td>
<td>Adjusts the width of the cubemap box.</td>
</tr>
</tbody>
</table>

## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cubemap_resolution</td>
<td>Sets the size of the cubemap.</td>
</tr>
<tr>
<td>preview_cubemap</td>
<td>Allows the cubemap to be visible in the level.</td>
</tr>
<tr>
<td>Outdoor Only</td>
<td>Prevents objects from rendering in a visible area.</td>
</tr>
<tr>
<td>Cast Shadow MinSpec</td>
<td>Enables an object to cast shadows using the selected quality setting and higher.</td>
</tr>
<tr>
<td>LodRatio</td>
<td>Defines the distance from the camera position that different level of detail (LOD) models for the object are used.</td>
</tr>
<tr>
<td>ViewDistanceMultiplier</td>
<td>Defines the distance from the camera position that an object is rendered.</td>
</tr>
<tr>
<td>HiddenInGame</td>
<td>Prevents objects from displaying in game mode.</td>
</tr>
</tbody>
</table>
## Lightning Arc 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 about the latest features, see the Amazon Lumberyard User Guide.

You can use the Lightning Arc entity to creates realistic electric arcing and sparking effects in your track view cinematics and levels.

### Material Setup

It is recommended to use a diffuse texture, transparency = 99, additive mode, with a slight glow, and using the Illum shader. The arc warps around the U coordinate and uses the V coordinate as a multi-frame animation.

### Lightning Arc Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>If set to true, it automatically starts sparking after jumping into the game.</td>
</tr>
<tr>
<td>ArcPreset</td>
<td>A valid preset must be given. This defines how the lightning arc looks.</td>
</tr>
<tr>
<td>Delay</td>
<td>Delay in seconds between sparks.</td>
</tr>
<tr>
<td>Delay Variation</td>
<td>Time randomization in seconds.</td>
</tr>
</tbody>
</table>

To set up the ArcPreset visual effect, open the `Libs\LightningArc\LightningArcEffects.xml` file and make desired changes. When finished, reload the `g_reloadGameFx` console variable.

### ArcPreset Parameters

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lightningDeviation</td>
<td>The smooth snaky effect given to the lightning in meters.</td>
</tr>
<tr>
<td>lightningFuzzyness</td>
<td>The noisy effect given to the lightning in meters.</td>
</tr>
<tr>
<td>lightningVelocity</td>
<td>After a spark is triggered, it starts to shift from its original position upwards.</td>
</tr>
<tr>
<td>branchMaxLevel</td>
<td>Should be kept at either 0 or 1, but either value can be used. However, it also allows child branches to strike out of the main</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>branchProbability</td>
<td>Probability that a child sparks from another beam segment. If set to 0, no branch is generated, 0.5 is a 50% probability of sparking a branch, 2.0 is a probability of sparking 2 per beam, and so on.</td>
</tr>
<tr>
<td>maxNumStrikes</td>
<td>Hard limit on the number of beam segments that can be generated regardless of previous parameters.</td>
</tr>
<tr>
<td>strikeTimeMin</td>
<td>Minimum time a spark is kept alive.</td>
</tr>
<tr>
<td>strikeTimeMax</td>
<td>Maximum time a spark is kept alive.</td>
</tr>
<tr>
<td>strikeFadeOut</td>
<td>When the spark dies, it takes this time to fade out into oblivion. It decreases beamSize to 0 instead of actually fading via transparency.</td>
</tr>
<tr>
<td>strikeNumSegments</td>
<td>Number of snaky segments generated.</td>
</tr>
<tr>
<td>strikeNumPoints</td>
<td>The number of actual segments generated is defined by strikeNumSegments<em>strikeNumPoint. When the code generates the geometry, it creates a camera/beam-aligned quad with exactly 2 triangles. This means that the number of triangles per strike is going to be strikeNumSegments</em>strikeNumPoint<em>2. Because maxNumStrikes is the hard limit of potential number of sparks active at any time, the potential number poly count of a given lightning effect is going to be strikeNumSegments</em>strikeNumPoint<em>2</em>maxNumStrike. However, remember that every time the LightningArc entity strikes, a new lightning effect is going to be triggered and therefore the total poly count of a given effect can go higher. The game has a internal hard limit for the total amount of lightning effects, lightning strikes, and poly count that cannot be surpassed; otherwise, geometry starts to disappear.</td>
</tr>
<tr>
<td>beamSize</td>
<td>Width of the beam being generated. Child beams have half the width.</td>
</tr>
<tr>
<td>beamTexTiling</td>
<td>Texture tiling depends on the world size of the actual beam being mapped. A value of 2.0 means the texture wraps around twice every meter. A value of 0.25 means the texture warps around every 4 meters. Only the U coordinate of the texture map is affected by this parameter.</td>
</tr>
<tr>
<td>beamTexShift</td>
<td>The U coordinate moves in a direction at this value's rate. While beamTexTiling only affects 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 multi-frame animation.</td>
</tr>
</tbody>
</table>

**Using Flow Graph**

The entity:LightningArc node is used for creating special arcing effects.
entity: LightningArc node I/O ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable/Disable</td>
<td>Allows to dynamically enable or disable the internal timer.</td>
</tr>
<tr>
<td>Strike</td>
<td>Allows to manually trigger a spark even when the entity is disabled. This allows synchronization of the spark effect with other level events.</td>
</tr>
<tr>
<td>EntityId</td>
<td>The entity that was last struck.</td>
</tr>
<tr>
<td>StrikeTime</td>
<td>The time the last spark takes to fade out.</td>
</tr>
</tbody>
</table>

Parameter Reload

Since the lightning effect is implemented using the Game Effects gem, it is possible to reload all parameters during runtime using the `g_reloadGameFx` console command.

Miscellaneous Entities

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 about the latest features, see the Amazon Lumberyard User Guide.

Miscellaneous entities are commonly used in level design.

The following area objects and entities can be accessed from the Misc button on the Objects tab of the Rollup Bar.

- CharAttachHelper
- Comment
- GravityVolume
- ReferencePicture
- SplineDistributor

CharAttachHelper

The CharAttachHelper object can be used to attach any arbitrary object to any bone of a character. The CharAttachHelper object must be linked to the target character, as well as the object to the CharAttachHelper. Use the Link Object button located in the toolbar to link objects.

Comment

The comment object allows the adding of comments anywhere inside a level. Comments can be used as a communication device if multiple people work on the same level.

To show comments in game, go to the Console window and type `cl_comment 1`.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CharsPerLine</td>
<td>Maximum number of characters per line of text</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Diffuse</td>
<td>Set the color of the text</td>
</tr>
<tr>
<td>Fixed</td>
<td>When using comments to indicate problems/bugs/issues in the level, this field can be used to mark them as “fixed”. The text and icon color changes to green</td>
</tr>
<tr>
<td>Hidden</td>
<td>Hides the text.</td>
</tr>
<tr>
<td>MaxDist</td>
<td>Maximum distance where the comment is shown. If camera is further than this, the comment is hidden</td>
</tr>
<tr>
<td>Size</td>
<td>Size of the text</td>
</tr>
<tr>
<td>Text</td>
<td>Text to display</td>
</tr>
</tbody>
</table>

**GravityVolume**

The GravityVolume entity can be used to create tunnels through which the player is getting pushed by an invisible force. It does so by modifying the global gravity variable so that the player stays afloat while maintaining momentum.

Place a GravityVolume entity in the level and in a similar way to placing out a road or river, draw the gravity volume out. Once you have your shape finished double-click the left mouse to finalize the shape.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radius</td>
<td>Defines the radius how wide the tube is.</td>
</tr>
<tr>
<td>Gravity</td>
<td>Defines how fast objects are getting pushed through the tube.</td>
</tr>
<tr>
<td>Falloff</td>
<td>Sets up how the gravity should be decreased at the edge of the tube.</td>
</tr>
<tr>
<td>Damping</td>
<td>Specifies the damping amount.</td>
</tr>
<tr>
<td>StepSize</td>
<td>Defines how fine the subdivision of the tube geometry segments should be.</td>
</tr>
<tr>
<td>DontDisableInvisible</td>
<td>Active this property so that invisible ones don't get disabled.</td>
</tr>
<tr>
<td>Enabled</td>
<td>Turns the gravity effect on/off.</td>
</tr>
</tbody>
</table>

**ReferencePicture**

The ReferencePicture object is used with the ReferenceImage shader and does not receive light or other shader information from within the level. It keeps the image at its pure source.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>The image file used as the reference picture.</td>
</tr>
</tbody>
</table>
SplineDistributor

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometry</td>
<td>This option specifies the geometry that needs to be used for the object.</td>
</tr>
<tr>
<td>Step Size</td>
<td>Sets the distance between each point along the spline. Smaller values increase the polygon count of the surface but also smooths out corners.</td>
</tr>
<tr>
<td>OutdoorOnly</td>
<td>When set, the object will not be rendered when inside a vis area.</td>
</tr>
<tr>
<td>RainOccluder</td>
<td>Set the brush to occlude rain, this works in conjunction with Rain Entity. If your level does contain rain, you should set this wisely, as there is a limit of 512 objects that can occlude at any given time.</td>
</tr>
<tr>
<td>SupportSecondVisArea</td>
<td>Normally, objects are considered to be in only one vis area. This option allows them to be added to multiple vis areas if their bounding box overlaps them, at the cost of some performance. Without this option, some large objects may not be displayed when viewed through portals in certain situations.</td>
</tr>
<tr>
<td>Hideable</td>
<td>When this option is set, AI will use this object as a hiding spot, using the specified hide point type.</td>
</tr>
<tr>
<td>LodRatio</td>
<td>Defines how far from the current camera position, the different Level Of Detail models for the object are used.</td>
</tr>
<tr>
<td>ViewDistanceMultiplier</td>
<td>Sets the distance from the current view at which the object renders.</td>
</tr>
<tr>
<td>NotTriangulate</td>
<td>Deprecated</td>
</tr>
<tr>
<td>NoStaticDecals</td>
<td>When this option is set, decals will not project onto the object.</td>
</tr>
<tr>
<td>RecvWind</td>
<td>When this option is set, the object will be affected by the level wind.</td>
</tr>
<tr>
<td>Occluder</td>
<td>Used for the construction of a level occlusion mesh.</td>
</tr>
</tbody>
</table>

Particle Entities

Particle effect entities act as a container for particle effects and can be attached to any object using the link feature. Particle entity properties become available after dragging a particle effect into a level or by selecting it.

Entity Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Sets the initially active or inactive. Can be toggled in the editor for testing.</td>
</tr>
<tr>
<td>AttachForm</td>
<td>If AttachType is not empty, this property determines where particles emit from the attached geometry. Set to Vertices, Edges, Surface, or Volume.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AttachType</td>
<td>If this entity is attached to a parent entity, this field can be used to cause particles to emit from the entity's geometry. Set to BoundingBox, Physics, or Render to emit from the applicable geometry.</td>
</tr>
<tr>
<td>CountPerUnit</td>
<td>If AttachType is not empty, this multiples the particle count by the &quot;extent&quot; of the attached geometry. Depending on AttachForm, the extent is either total vertex count, edge length, surface area, or volume.</td>
</tr>
<tr>
<td>CountScale</td>
<td>Multiplies the particle counts of the entire emitter.</td>
</tr>
<tr>
<td>ParticleEffect</td>
<td>Use to generate the following effects:</td>
</tr>
<tr>
<td>Prime</td>
<td>If true, and the assigned ParticleEffect is immortal, causes the emitter to start &quot;primed&quot; to its equilibrium state, rather than starting up from scratch. Very useful for placed effects such as fires or waterfalls, which are supposed to be already running when the level starts. Applies only to immortal, not mortal effects.</td>
</tr>
<tr>
<td>PulsePeriod</td>
<td>If not 0, restarts the emitter repeatedly at this time interval. Should be used to create emitters that pulse on and off at somewhat large intervals, a second or so. Do not set a low value such as 0.1 to try to make an instant effect into a continuous one. Make sure the actual library effect is set Continuous and has an appropriate Count.</td>
</tr>
<tr>
<td>RegisterByBBox</td>
<td>Uses the emitter's (automatically computed) bounding box to determine which vis areas it is visible in. If this is disabled (the default), the emitter's origin alone determines vis area membership, as the bounding box is hard to exactly control by the designer.</td>
</tr>
<tr>
<td>Scale</td>
<td>Multiplies the overall size and velocity of the entire emitter.</td>
</tr>
<tr>
<td>SpeedScale</td>
<td>Multiplies the particle emission speed of the entire emitter.</td>
</tr>
<tr>
<td>Strength</td>
<td>Used by effect parameters to modify their value. If a parameter has a Strength Over Emitter Life curve, and the emitter entity's Strength property is not negative, then Strength will be used as input to this curve.</td>
</tr>
<tr>
<td>TimeScale</td>
<td>Multiplies the elapsed time used to simulate the emitter. Less than 1 achieves a show-motion effect.</td>
</tr>
<tr>
<td>EnableAudio</td>
<td>Toggles sound emission on any sub-effects with an Audio parameter set.</td>
</tr>
</tbody>
</table>

### Physics Entities

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 about the latest features, see the Amazon Lumberyard User Guide.
Physics entities are 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.

The following entities can be accessed by clicking Entity, then expanding Physics on the Objects tab of the Rollup Bar.

- AnimObject
- BasicEntity
- Constraint
- DeadBody
- GravityBox
- GravitySphere
- GravityValve
- LivingEntity
- ParticlePhysics
- RigidBodyEx
- Wind
- WindArea

**AnimObject**

An AnimObject extends the functionality of a BasicEntity by the ability of playing pre-baked animations and physicalizing parts of the object afterwards.

**AnimObject Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActivatePhysicsDist</td>
<td>Used for objects with pre-baked physical animations (requires Articulated to be on and ActivatePhysicsThreshold to be greater than 0). Specifies the distance from the pivot after which parts automatically detach themselves from the animation and become fully physicalized. 0 disables distance-based detachment.</td>
</tr>
<tr>
<td>ActivatePhysicsThreshold</td>
<td>Greater than 0 values are used for objects with pre-baked physical animations (requires Articulated to be on). Specifies the amount of force (in fractions of gravity) that needs to be exerted on a part for it to become detached and fully controlled by the physics.</td>
</tr>
<tr>
<td>CanTriggerAreas</td>
<td>Triggers when this entity enters/exits. Only applicable to AreaTriggers; ProximityTriggers triggers regardless.</td>
</tr>
<tr>
<td>DmgFactorWhenCollidingAI</td>
<td>Multiplier applied when dealing damage to AI.</td>
</tr>
<tr>
<td>Faction</td>
<td>Entity faction.</td>
</tr>
<tr>
<td>InteractLargeObject</td>
<td>Players can trigger large object interactions (such as grab and kick) with the entity.</td>
</tr>
<tr>
<td>MissionCritical</td>
<td>Entity is not be hidden by explosions.</td>
</tr>
<tr>
<td>Model</td>
<td>Defines the CGA model to be used.</td>
</tr>
<tr>
<td>Pickable</td>
<td>Defines whether or not the object can be picked up.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SmartObjectClass</td>
<td>Specifies the smart object type of the object.</td>
</tr>
<tr>
<td>Usable</td>
<td>Defines whether or not the object can be used.</td>
</tr>
<tr>
<td>UseMessage</td>
<td>The message displayed when the object is in the crosshairs for use.</td>
</tr>
<tr>
<td><strong>Animation</strong></td>
<td></td>
</tr>
<tr>
<td>Animation</td>
<td>Defines the animation to be played.</td>
</tr>
<tr>
<td>Loop</td>
<td>Defines whether the animation is looped.</td>
</tr>
<tr>
<td>PhysicalizeAfterAnimation</td>
<td>Defines whether the object is physicalized after the animation has reached its end.</td>
</tr>
<tr>
<td>playerAnimationState</td>
<td>If set, the animation plays immediately.</td>
</tr>
<tr>
<td>Playing</td>
<td>If set, the animation plays immediately.</td>
</tr>
<tr>
<td>Speed</td>
<td>Playback speed of the animation sequence.</td>
</tr>
<tr>
<td><strong>Health</strong></td>
<td></td>
</tr>
<tr>
<td>Invulnerable</td>
<td>Object does not receive damage, but registers &quot;Hit&quot; output when applicable.</td>
</tr>
<tr>
<td>MaxHealth</td>
<td>Health of the entity, how much damage can it take before being considered &quot;Dead&quot; and triggering the output.</td>
</tr>
<tr>
<td>OnlyEnemyFire</td>
<td>Takes damage from enemy (Faction-based) fire, only if a faction is set.</td>
</tr>
<tr>
<td><strong>MultiplayerOptions</strong></td>
<td></td>
</tr>
<tr>
<td>Networked</td>
<td>Physics is simulated on the server and serialized over the network; otherwise, simulated on the client.</td>
</tr>
<tr>
<td><strong>Physics</strong></td>
<td></td>
</tr>
<tr>
<td>Articulated</td>
<td>Physicalizes the character as an articulated physical entity (i.e., with bendable joints).</td>
</tr>
<tr>
<td>Density</td>
<td>Can be used instead of Mass (if mass is -1) to set the density of each node.</td>
</tr>
<tr>
<td>Mass</td>
<td>The overall mass for the entire model.</td>
</tr>
<tr>
<td>Physicalize</td>
<td>Selects whether or not the model can become physicalized.</td>
</tr>
<tr>
<td>PushableByPlayers</td>
<td>Allows the object to be pushed by players.</td>
</tr>
<tr>
<td>RigidBody</td>
<td>If deselected, the object is static. Pre-baked physics objects must have it selected.</td>
</tr>
</tbody>
</table>

**BasicEntity**

A BasicEntity provides the simplest way of controlling objects physically. Once a model has been set, several properties can be set, defining its physical behavior. It is possible to specify either density or mass...
of the object. If one is specified, the other one must be set to a negative value (-1, or -0.01). Mass and
density affect the way objects interact with other objects and float in the water (they sink if their density
is more than that of the water). A zero-mass rigid body (with both mass and density 0) is a special case
which means an "animated" rigid body (moved from outside the physics system).

The difference from a static entity is that the physics is aware that this object is actually dynamic,
although it cannot simulate it directly. Note that both values describe the same physical property. When
you specify mass, density is computed automatically, and vice versa. The relationship mass = density x
volume is used. These computations imply that the object is solid. If a box is used to model an empty
crate, one can assume that its density is a weighted average between wood density and inside air density.

### BaseEntity Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanTriggerAreas</td>
<td>Areas trigger when this entity enters/exits them. Only applicable to AreaTriggers; ProximityTriggers trigger regardless.</td>
</tr>
<tr>
<td>DmgFactorWhenCollidingAI</td>
<td>Multiplier applied when dealing damage to AI.</td>
</tr>
<tr>
<td>Faction</td>
<td>Entity faction.</td>
</tr>
<tr>
<td>InteractLargeObject</td>
<td>Players can trigger large object interactions (such as grab and kick) with the entity.</td>
</tr>
<tr>
<td>MissionCritical</td>
<td>Entity is not be hidden by explosions. The threshold for hiding/removal is defined via the console variable g_ec_removeThreshold which is set to 20 by default. If an explosion occurs and more than 20 entities are hit by it, it keeps 20 and hides the rest for better performance. See GameRulesClientServer.cpp for more information.</td>
</tr>
<tr>
<td>Model</td>
<td>Defines the model to be used.</td>
</tr>
<tr>
<td>Pickable</td>
<td>Players can grab or pick up the object.</td>
</tr>
<tr>
<td>SmartObjectClass</td>
<td>Can be used to define AI interaction capabilities on code-side.</td>
</tr>
<tr>
<td>Usable</td>
<td>Entity is usable by players.</td>
</tr>
<tr>
<td>UseMessage</td>
<td>If useable is true, this message is displayed when players are in range. Can be a localized string such as @use_object.</td>
</tr>
<tr>
<td>Health</td>
<td></td>
</tr>
<tr>
<td>Invulnerable</td>
<td>Object does not receive damage, but registers &quot;Hit&quot; output when applicable.</td>
</tr>
<tr>
<td>MaxHealth</td>
<td>Health of the entity, how much damage can it take before being considered &quot;Dead&quot; and triggering the output.</td>
</tr>
<tr>
<td>OnlyEnemyFire</td>
<td>Takes damage from enemy (faction-based) fire, only if a faction is set.</td>
</tr>
<tr>
<td>MultiplayerOptions</td>
<td></td>
</tr>
<tr>
<td>Networked</td>
<td>Physics is simulated on the server and serialized over the network; otherwise, simulates on the client.</td>
</tr>
<tr>
<td>Physics</td>
<td></td>
</tr>
</tbody>
</table>
### Property Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Density</td>
<td>(= \text{Mass} / \text{Volume}) Density affects the way objects interact with other objects and float in the water (they sink if their density is more than that of the water). Note that both density and mass can be overridden in the asset file.</td>
</tr>
<tr>
<td>Mass</td>
<td>(= \text{Density} \times \text{Volume}) Mass is the weight of the object (the density of the object multiplied by its volume).</td>
</tr>
<tr>
<td>Physicalize</td>
<td>If false, the object is not taken into account by physics.</td>
</tr>
<tr>
<td>PushableByPlayers</td>
<td>If true, the player pushes the object by walking/running into it.</td>
</tr>
<tr>
<td>RigidBody</td>
<td>False means a static entity, true - a simulated rigid body. Note that a rigid body can still behave like a static entity if it has mass 0 (set either explicitly or by unchecking RigidBodyActive). The main difference between these rigid bodies and pure statics is that the physics system knows that they can be moved by some other means (such as a track view sequence) and expects them to do so. This means that objects that are supposed to be externally animated should be mass-0 rigid bodies in order to interact properly with pure physicalized entities.</td>
</tr>
</tbody>
</table>

### Constraint

A constraint entity can create a physical constraint between two objects. The objects are selected automatically during the first update, by sampling the environment in a sphere around the constraint object's world position with a specified radius. The “first” object (the one that will own the constraint information internally) is the lightest among the found objects, and the second is the second lightest (static objects are assumed to have infinite mass, so a static object is always heavier than a rigid body).

Constraints operate in a special “constraint frame.” It can be set to be either the frame of the first constraint object (if UseEntityFrame is checked), or the frame of the constraint entity itself. In that frame, the constraint can operate either as a hinge around the x-axis, or as a ball-in-a-socket around y- and z-axes (that is, with the x-axis as the socket's normal). If x limits are set to a valid range \((\text{max}>\text{min})\) and the yz limits are identical (such as both ends are 0), it is the former and, if the yz limits are set and not x limits, it’s the latter. If all limits are identical (remain 0, for instance), the constraint operates in a 3 degrees of freedom mode (does not constrain any rotational axes). If all limits are set, no axes are locked initially, but there are rotational limits for them.

### Constraint Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>damping</td>
<td>Sets the strength of the damping on an object's movement. Most objects can work with 0 damping; if an object has trouble coming to rest, try values like 0.2-0.3. Values of 0.5 and higher appear visually as overdamping. Note that when several objects are in contact, the highest damping is used for the entire group.</td>
</tr>
<tr>
<td>max_bend_torque</td>
<td>The maximum bending torque (Currently it's only checked against for hinge constraints that have reached one of the x limits).</td>
</tr>
<tr>
<td>max_pull_force</td>
<td>Specifies the maximum stretching force the constraint can withstand.</td>
</tr>
</tbody>
</table>
Physics Entities

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoSelfCollisions</td>
<td>Disables collision checks between the constrained objects (To be used if the constraint is enough to prevent inter-penetrations).</td>
</tr>
<tr>
<td>radius</td>
<td>Defines spherical area to search for attachable objects.</td>
</tr>
<tr>
<td>UseEntityFrame</td>
<td>Defines whether to use the first found object or the constraint itself as a constraint frame.</td>
</tr>
</tbody>
</table>

**Limits**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_max</td>
<td>If set greater than x_min, the constraint only rotates the object along its x-axis within the defined angle.</td>
</tr>
<tr>
<td>x_min</td>
<td>See x_max.</td>
</tr>
<tr>
<td>yz_max</td>
<td>If set greater than yz_min, the constraint only rotates the object along its yz-axis within the defined angle.</td>
</tr>
<tr>
<td>yz_min</td>
<td>See yz_max.</td>
</tr>
</tbody>
</table>

**DeadBody**

A DeadBody entity can ragdollize characters assigned to it. As soon as a character is intended not to act any more, but to only react passively on external impacts, as if it were dead, this physical entity provides the necessary model.

A typical usage is to create the entity as non-resting, simulate it in the editor, and then save the settled physics state. Note that the entity does not react to collisions with the player, bullets, or explosions.

**DeadBody Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CollidesWithPlayers</td>
<td>Defines whether the ragdoll of the entity may collide with the player (does not override the non-interactive ragdoll legal restriction)</td>
</tr>
<tr>
<td>ExtraStiff</td>
<td>Uses the main solver to apply stiffness instead of joint springs. It can handle a lot higher stiffness values, but the downside is that the same stiffness is applied to all joint axes, including locked and limited ones.</td>
</tr>
<tr>
<td>lying_damping</td>
<td>(0..1..10) Defines damping in the &quot;lying&quot; mode (which is when the ragdoll has enough contacts with the ground). Note that this is an overall damping, and there also exist per-joint dampings, set based on the asset.</td>
</tr>
<tr>
<td>mass</td>
<td>The mass of the object.</td>
</tr>
<tr>
<td>MaxTimeStep</td>
<td>As with other entities, decreasing it makes the simulation more stable, but makes this entity and all entities it contacts with more expensive to simulate. Can be especially useful when higher stiffness is needed.</td>
</tr>
<tr>
<td>Model</td>
<td>Character model to be physicalized.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NoFriendlyFire</td>
<td>If set, the entity does not react on bullet impacts from friendly units.</td>
</tr>
<tr>
<td>PoseAnim</td>
<td>Allows to use the first frame of the specified animation as an initial pose.</td>
</tr>
<tr>
<td>PushableByPlayers</td>
<td>If set, the entity does not react on bullet impacts from friendly units.</td>
</tr>
<tr>
<td>PushableByPlayers</td>
<td>See BasicEntity (does not override the non-interactive ragdoll legal restriction)</td>
</tr>
<tr>
<td>Resting</td>
<td>If set, object do not spawn in a physically 'awake' state. Instead it waits until physically interacted with first.</td>
</tr>
<tr>
<td>SmartObjectClass</td>
<td>Specifies the smart object type of the object.</td>
</tr>
<tr>
<td>Stiffness</td>
<td>Stiffness with which the ragdoll tries to maintain the original pose (set either in the model or from PoseAnim). For SDK character values around 2000 are practical. Higher values can lead to stability issues, which can be overcome by either decreasing MaxTimeStep (which makes it more expensive to simulate), or using ExtraStiff mode.</td>
</tr>
<tr>
<td>Buoyancy</td>
<td></td>
</tr>
<tr>
<td>water_damping</td>
<td>A cheaper alternative/addition to water resistance (applies uniform damping when in water). Sets the strength of the damping on an object's movement as soon as it is situated underwater. Most objects can work with 0 damping; if an object has trouble coming to rest, try values like 0.2-0.3. Values of 0.5 and higher appear visually as overdamping. Note that when several objects are in contact, the highest damping is used for the entire group.</td>
</tr>
</tbody>
</table>
| water_density          | Can be used to override the default water density (1000). Lower values assume that the body is floating in the water that's less dense than it actually is, and thus it sinks easier.  

(100..1000) This parameter could be used to specify that the object's physical geometry can leak. For instance, ground vehicles usually have quite large geometry volumes, but they are not waterproof, thus Archimedean force acting on them is less than submerged_volume 1000 (with 1000 being the actual water density).

Decreasing per-object effective water density allows such objects to sink while still having large-volume physical geometry.

Important note: If you are changing the default value (1000), it is highly recommended that you also change water_resistance in the same way (a rule of thumb might be to always keep them equal).
Property | Description
--- | ---
waters\_resistance | Can be used to override the default water resistance (1000). Sets how strongly the water affects the body (this applies to both water flow and neutral state).

(0..2000) Water resistance coefficient. If non-0, precise water resistance is calculated. Otherwise only water\_damping (proportional to the submerged volume) is used to uniformly damp the movement. The former is somewhat slower, but not prohibitively, so it is advised to always set the water resistance.

Although water resistance is not too visible on a general object, setting it to a suitable value prevents very light objects from jumping in the water, and water flow affects things more realistically.

Note that water damping is used regardless of whether water resistance is 0, so it is better to set damping to 0 when resistance is turned on.

---

**GravitySphere**

A GravitySphere is a spherical area, which replaces the gravitational parameters of the environment. Objects reaching this area moved along the entities' Gravity vector and their own physical impact can be damped by a certain factor.

**GravitySphere Properties**

| Property | Description |
--- | --- |
Active | Defines whether the entity affects its environment. |
Damping | Damps physical impact of entities inside the sphere. |
Radius | Size of the sphere. |
Gravity | x,y, z vector of the gravity applied to objects within the sphere. |

---

**GravityValve**

A GravityValve entity performs an additional gravity into an upwards showing direction, relative to the entity.

**GravityValve Properties**

| Property | Description |
--- | --- |
Active | Defines whether the entity affects its environment. |
Radius | Size of the affected area. |
Strength | Gravitational force. |
Wind

A wind entity is used to simulate wind in a local position. This should not be used to create the global wind in your level.

**Wind Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FadeTime</td>
<td>The time the wind entity uses to fade between disabled and enabled states.</td>
</tr>
<tr>
<td>vVelocity</td>
<td>(x,y,z) vector sets the direction and strength of the wind.</td>
</tr>
</tbody>
</table>

WindArea

A WindArea simulates air moving with an arbitrary speed in a specific direction. It affects the flow direction of all objects and aero-form substances within the defined area, as well as vegetation bending depending on density and resistance values. If no direction is set, the wind-source moves omnidirectionally from the center of the WindArea.

**WindArea Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>Defines whether wind is blowing or not.</td>
</tr>
<tr>
<td>AirDensity</td>
<td>Causes physicalized objects moving through the air to slow down, if &gt; 0.</td>
</tr>
<tr>
<td>AirResistance</td>
<td>Causes very light physicalized objects to experience a buoyancy force, if &gt; 0.</td>
</tr>
<tr>
<td>Ellipsoidal</td>
<td>Forces an ellipsoidal falloff.</td>
</tr>
<tr>
<td>FalloffInner</td>
<td>Distance after which the distance-based falloff begins.</td>
</tr>
<tr>
<td>Speed</td>
<td>Wind-speed in units per second.</td>
</tr>
<tr>
<td>Dir XYZ</td>
<td>(x,y,z) vector of normalized wind direction.</td>
</tr>
<tr>
<td>Size XYZ</td>
<td>(x,y,z) vector of affected area.</td>
</tr>
</tbody>
</table>

Useful Console Variables

The following console variables are useful for debugging physics entity issues:

```
_p_draw_helpers
Same as _p_draw_helpers_num, but encoded in letters
Usage [Entity_Types]_[Helper_Types] - [t|s|x|R|l|i|g|a|y|e]_[g|c|b|l|t(#)]
Entity Types:
t - show terrain
s - show static entities
r - show sleeping rigid bodies
R - show active rigid bodies
l - show living entities
```
Rain Entity

You can use the Rain entity to add realistic rain effects to your level.

Entity Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount</td>
<td>Sets the amount of rain and rain effects in a level. AttenAmount is</td>
</tr>
<tr>
<td></td>
<td>multiplied by the amount, and is used to set the current amount.</td>
</tr>
<tr>
<td>DiffuseDarkening</td>
<td>Modifies the albedo of the rain effect, such as for horizontal water</td>
</tr>
<tr>
<td></td>
<td>puddles.</td>
</tr>
<tr>
<td>DisableOcclusion</td>
<td>Blocks rain for selected objects in your level. Don't select for objects</td>
</tr>
<tr>
<td></td>
<td>that are protected (under cover) from rain.</td>
</tr>
<tr>
<td>Enabled</td>
<td>Enables or disables the rain effects.</td>
</tr>
<tr>
<td>FakeGlossiness</td>
<td>Sets the amount of glossiness for wet surfaces.</td>
</tr>
<tr>
<td>FakeReflectionsAmount</td>
<td>Sets the amount of reflection from wet surfaces.</td>
</tr>
<tr>
<td>IgnoreVisAreas</td>
<td>Renders rain even when player is inside a vis area.</td>
</tr>
<tr>
<td>PuddlesAmount</td>
<td>Sets the depth and brightness of water puddles generated by the rain.</td>
</tr>
<tr>
<td>PuddlesMaskAmount</td>
<td>Sets the strength of the water puddle mask to balance different puddle</td>
</tr>
<tr>
<td></td>
<td>results.</td>
</tr>
<tr>
<td>PuddlesRipplesAmount</td>
<td>Sets the strength and frequency of ripples in water puddles.</td>
</tr>
<tr>
<td>Radius</td>
<td>Sets the coverage area of rain around the entity.</td>
</tr>
<tr>
<td>RainDropsAmount</td>
<td>Sets the amount of rain drops that can be seen in the air.</td>
</tr>
<tr>
<td>RainDropsLighting</td>
<td>Sets the brightness or backlighting of the rain drops.</td>
</tr>
<tr>
<td>RainDropsSpeed</td>
<td>Sets the speed at which rain drops travel.</td>
</tr>
<tr>
<td>SplashesAmount</td>
<td>Modifies the strength of the splash effect.</td>
</tr>
</tbody>
</table>

p_debug_joints

If set, breakable objects log tensions at the weakest spots.
Render Entities

You can use the following Render entities in your level.

FogVolume Entity

Entity Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>If true, fog volume will be enabled.</td>
</tr>
<tr>
<td>Color</td>
<td>Specifies the RGB diffuse color of the fog volume</td>
</tr>
<tr>
<td>DensityOffset</td>
<td>Used in conjunction with the <strong>GlobalDensity</strong> parameter to offset the density.</td>
</tr>
<tr>
<td>FallOffDirLati</td>
<td>Controls the latitude falloff direction of the fog. A value of 90° means the falloff direction is upwards.</td>
</tr>
<tr>
<td>FallOffDirLong</td>
<td>Controls the longitude falloff direction of the fog, where 0° represents east. Rotation is counterclockwise.</td>
</tr>
<tr>
<td>FallOffScale</td>
<td>Scales the density distribution along the falloff direction. Higher values make the fog fall off more rapidly and generate thicker fog layers along the negative falloff direction.</td>
</tr>
<tr>
<td>FallOffShift</td>
<td>Controls how much to shift the fog density distribution along the falloff direction in world units (m). Positive values move thicker fog layers along the falloff direction into the fog volume.</td>
</tr>
<tr>
<td>GlobalDensity</td>
<td>Controls the density of the fog. The higher the value the more dense the fog.</td>
</tr>
<tr>
<td>HDRDynamic</td>
<td>Specifies how much brighter than the default white (RGB 255,255,255) the fog is.</td>
</tr>
<tr>
<td>NearCutoff</td>
<td>Stops rendering the object depending on camera distance to object.</td>
</tr>
<tr>
<td>SoftEdges</td>
<td>Factor used to soften the edges of the fog volume when viewed from outside. A value of 0.0 produces hard edges. Increasing this value up to 1.0 gradually softens the edges. This property currently has no effect on box type fog volumes as specified in the <strong>VolumeType</strong> parameter.</td>
</tr>
<tr>
<td>UseGlobalFogColor</td>
<td>If selected, ignores the <strong>Color</strong> parameter and uses the global (Time Of Day) fog color instead.</td>
</tr>
<tr>
<td>VolumeType</td>
<td>Produces a box volume for values above 1.0 or a spherical volume for lower values.</td>
</tr>
<tr>
<td>Size x, y, z</td>
<td>Specifies the height, width, and depth of the fog volume in meters.</td>
</tr>
</tbody>
</table>

River Entity

You can customize your rivers with a number of different parameters. Many of the settings are the same as those of the **WaterVolume Shader**.
**Note**
The Speed parameter listed below specifies the speed at which objects float down the river. The speed of the river itself is specified using the **Flow speed** parameter for the **WaterVolume Shader**.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Width</strong></td>
<td>Sets the width of the river. This is set much wider than the actual river (water) width, as the complete river is defined by the river bed and surrounding terrain.</td>
</tr>
<tr>
<td><strong>BorderWidth</strong></td>
<td>Used in conjunction with <strong>Align Height Map</strong>, creates a smooth edge for the river bed geometry if this value is greater than the <strong>Width</strong> value.</td>
</tr>
<tr>
<td><strong>StepSize</strong></td>
<td>Sets the distance between each point along the river spline. Smaller values increase the polygon count of the river surface but also smooths out corners.</td>
</tr>
<tr>
<td><strong>ViewDistanceMultiplier</strong></td>
<td>Sets the distance from the current view at which the river renders.</td>
</tr>
<tr>
<td><strong>TileLength</strong></td>
<td>Length of the river texture. Use in conjunction with <strong>StepSize</strong> to avoid stretching textures.</td>
</tr>
<tr>
<td><strong>Depth</strong></td>
<td>Sets the depth of the river.</td>
</tr>
<tr>
<td><strong>Speed</strong></td>
<td>Defines how fast physicalized objects move along the river. Use negative values to move in the opposite direction.</td>
</tr>
<tr>
<td><strong>UScale</strong></td>
<td>Sets the texture tiling on the U axis.</td>
</tr>
<tr>
<td><strong>VScale</strong></td>
<td>Sets the texture tiling on the V axis.</td>
</tr>
<tr>
<td><strong>Caustics</strong></td>
<td>Enables optical caustics effects.</td>
</tr>
<tr>
<td><strong>CausticIntensity</strong></td>
<td>Scales the intensity of the caustics for the water surface normals.</td>
</tr>
<tr>
<td><strong>CausticTiling</strong></td>
<td>Scales the caustic tiling applied to the water surface normals. It allows the scaling of caustics independently from the surface material.</td>
</tr>
<tr>
<td><strong>CausticHeight</strong></td>
<td>Sets the height above the water surface at which caustics become visible. Use this to make caustics appear on overhanging landforms or vegetation and other nearby objects.</td>
</tr>
</tbody>
</table>

### Road Entity

You can modify any of several road parameters to customize your road.

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Width</strong></td>
<td>Width of the road.</td>
</tr>
<tr>
<td><strong>BorderWidth</strong></td>
<td>Used in conjunction with <strong>Align Height Map</strong>, creates a smooth edge for the road if this value is greater than the <strong>Width</strong> value.</td>
</tr>
</tbody>
</table>

1031
### Parameter | Description
---|---
**StepSize** | Sets the distance between each point along the road spline. Smaller values increase the polygon count for the road surface but also smooths out corners.

**ViewDistanceMultiplier** | Specifies the distance at which the road renders.

**TileLength** | Length of the road texture. Used in conjunction with **StepSize** to avoid stretching textures.

**SortPriority** | Determines the rendering order. Higher values are rendered above lower values.

**IgnoreTerrainHoles** | If enabled, renders the road texture over holes created with the terrain Holes brush.

---

**Rope Entity**

The Rope entity is used to create realistic ropes in your level.

**Parameters**

| Parameter | Description |
---|---|
**Radius** | The radius, or thickness, of the rope. |
**Smooth** | Defines if the rope will be smoothed out or not. |
**Num Segments** | The number of segments of geometry used in the rope along its length. |
**Num Sides** | The number of sides around the circumference of the rope. 4 sides would make it a diamond shaped tube, 8 sides would make it much smoother, etc. |
**Texture U Tiling** | Texture tiling in the U direction. |
**Texture V Tiling** | Texture tiling in the V direction. |
**CastShadows** | Enable shadow casting from the rope. |
**Bind Ends Radius** | Specifies whether the ends will be automatically attached. |
**Bind Radius** | The environment around the ends of the rope will be tested using a box of this radius to find places for the rope to attached to. Note that if bind radius is greater than 0.05 the ends are snapped to the colliding surface. |

**Physics Params**

| Parameter | Description |
---|---|
**Subdivide** | Maximum number of subdivided vertices per segment. |
**Max Subdiv Verts** | Maximum number of subdivided vertices per segment. |
**Physical Segments** | Number of rope segments in physics (can be different from the number of segments used for rendering). For colliding ropes, make sure that there are enough physical segments so that segment |
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>length is at least two times smaller than the dimensions of the objects the rope collides with.</td>
</tr>
<tr>
<td>Tension</td>
<td>Specifies tension in the original state. A positive value will cause the rope ends to pull together, negative will add slack to the rope (-0.02 is a good starting point for experiments).</td>
</tr>
<tr>
<td>Friction</td>
<td>The friction effective in a non-strained mode. In a strained mode with dynamic tessellation, this that prevents the rope from slipping until it tilts too much.</td>
</tr>
<tr>
<td>Wind</td>
<td></td>
</tr>
<tr>
<td>Wind Variation</td>
<td>How much the wind varies. Basically a randomization multiplier on top of the base Wind XYZ values.</td>
</tr>
<tr>
<td>Air Resistance</td>
<td>Must be set in order for global environment wind to take effect. Not necessary for simulated Wind XYZ values.</td>
</tr>
<tr>
<td>Water Resistance</td>
<td>How the rope interacts with water effectively damping when under water.</td>
</tr>
<tr>
<td>Check Collisions</td>
<td>Ignore collisions from other objects.</td>
</tr>
<tr>
<td>Ignore Attachment Collisions</td>
<td>Ignore collisions with the object it is attached to.</td>
</tr>
<tr>
<td>Ignore Player Collisions</td>
<td>Ignore collisions with players.</td>
</tr>
<tr>
<td>Non-shootable</td>
<td>Rope cannot be broken by shooting. Rope will still react to physical impulses from bullets.</td>
</tr>
<tr>
<td>Disabled</td>
<td>Simulation is completely disabled.</td>
</tr>
<tr>
<td>StaticAttachStart</td>
<td>Attach start point to the level.</td>
</tr>
<tr>
<td>StaticAttachEnd</td>
<td>Attach end point to the level.</td>
</tr>
<tr>
<td>Advanced</td>
<td></td>
</tr>
<tr>
<td>Mass</td>
<td>This affects how strongly the rope will react to bullet hits. When interacting with solid physicalized objects, it is always treated as weightless.</td>
</tr>
<tr>
<td>Friction Pull</td>
<td>The friction effective in a non-strained mode. In a strained mode with dynamic tessellation, this that prevents the rope from slipping until it tilts too much.</td>
</tr>
<tr>
<td>Max Force</td>
<td>The rope will detach itself when this strain limit is breached.</td>
</tr>
<tr>
<td>Solver Iterations</td>
<td>Ropes with very large segment counts (40+) might need this increased (values up to 10k are still viable).</td>
</tr>
<tr>
<td>Max Timestamp</td>
<td>Sets the maximum time step the entity is allowed to make (defaults to 0.01). Smaller time steps increase stability (can be required for long and thin objects, for instance), but are more expensive. Each time the physical world is requested to make a step, the objects that have their maxsteps smaller than the requested one slice the big step into smaller chunks and perform several substeps. If several objects are in contact, the smallest max_time_step is used.</td>
</tr>
</tbody>
</table>
### Snow Entity

You can use the Snow entity to add realistic snow effects to your level.

#### Entity Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Select to enable snow.</td>
</tr>
<tr>
<td>Radius</td>
<td>Sets the coverage area of snow on the ground. Has no effect on the distance that snow in the air spawns at.</td>
</tr>
<tr>
<td>Brightness</td>
<td>The brightness of snowflakes in the air.</td>
</tr>
<tr>
<td>GravityScale</td>
<td>Controls how fast snow falls.</td>
</tr>
</tbody>
</table>
Tornado Entity
You can create realistic-looking tornadoes in your level.

**Entity Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttractorImpulse</td>
<td>The gravitational pull of the tornado on nearby objects.</td>
</tr>
<tr>
<td>CloudHeight</td>
<td>The height of the cloud above the tornado.</td>
</tr>
<tr>
<td>FunnelEffect</td>
<td>Specifies the particular particle effect.</td>
</tr>
<tr>
<td>Radius</td>
<td>Radius of the tornado's influence.</td>
</tr>
<tr>
<td>SpinImpulse</td>
<td>The rotational speed of the tornado.</td>
</tr>
<tr>
<td>UpImpulse</td>
<td>The upward speed of the tornado.</td>
</tr>
<tr>
<td>WanderSpeed</td>
<td>The speed that the tornado is moving along the ground.</td>
</tr>
</tbody>
</table>

Trigger Entities
There are two Trigger entities you can use in your level.

**AreaTrigger Entity**

**Entity Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trigger-proximity</td>
<td>Turns the entity on or off.</td>
</tr>
<tr>
<td>InVehicleOnly</td>
<td>Sets up that the trigger can only be activated when player is inside vehicle.</td>
</tr>
<tr>
<td>OnlyLocalPlayer</td>
<td>Sets the trigger to be only triggerable by the local player entity.</td>
</tr>
<tr>
<td>OnlyPlayers</td>
<td>Sets the trigger to be only triggerable by players entities.</td>
</tr>
</tbody>
</table>
### Property Reference

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PlaySequence</td>
<td>Plays the sequence with the name specified in here.</td>
</tr>
<tr>
<td>ScriptCommand</td>
<td>Executes a script command when the trigger has been activated.</td>
</tr>
<tr>
<td>TriggerOnce</td>
<td>Disables the trigger after it has been triggered once.</td>
</tr>
</tbody>
</table>

### MultiplayerOptions

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Networked</td>
<td>If true physics will be simulated on the server and serialized over the network, otherwise they will be simulated on the client.</td>
</tr>
</tbody>
</table>

### ProximityTrigger Entity

#### Entity Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActivateWithUseButton</td>
<td>Specifies if the trigger is activated by pressing use.</td>
</tr>
<tr>
<td>DimX</td>
<td>Specifies how big the trigger is (x-axis).</td>
</tr>
<tr>
<td>DimY</td>
<td>Specifies how big the trigger is (y-axis).</td>
</tr>
<tr>
<td>DimZ</td>
<td>Specifies how big the trigger is (z-axis).</td>
</tr>
<tr>
<td>Enabled</td>
<td>Specifies if the trigger can be activated or not.</td>
</tr>
<tr>
<td>EnterDelay</td>
<td>Sets up a delay (in seconds) before the enter node of the trigger is activated.</td>
</tr>
<tr>
<td>ExitDelay</td>
<td>Sets up a delay (in seconds) before the exit node of the trigger is activated.</td>
</tr>
<tr>
<td>InVehicleOnly</td>
<td>Sets up that the trigger can only be activated when player is inside vehicle.</td>
</tr>
<tr>
<td>OnlyAI</td>
<td>Sets the trigger to be only triggerable by AI entities.</td>
</tr>
<tr>
<td>OnlyMyPlayer</td>
<td>Sets the trigger to be only triggerable by the local player.</td>
</tr>
<tr>
<td>OnlyOneEntity</td>
<td>Sets the trigger to be only triggerable by one entity. First one who triggers it has to leave it in order to be triggerable again.</td>
</tr>
<tr>
<td>OnlyPlayer</td>
<td>Sets the trigger to be only triggerable by player entities.</td>
</tr>
<tr>
<td>OnlySelectedEntity</td>
<td>Sets the trigger to be only triggerable by the entity with the name specified in this field. Wildcard matches can be used such as RigidbodyEx*, will allow all entities with that name, regardless of number suffix, etc.</td>
</tr>
<tr>
<td>OnlySpecialAI</td>
<td>Sets the trigger to be only triggerable by the special AI entities.</td>
</tr>
<tr>
<td>PlaySequence</td>
<td>Plays the sequence with the name specified in here.</td>
</tr>
<tr>
<td>RemoveOnTrigger</td>
<td>Similar to the deprecated &quot;KillOnTrigger&quot; param, if true, any entities (except player) which trigger this will be removed.</td>
</tr>
<tr>
<td>ScriptCommand</td>
<td>Executes a script command when the trigger has been activated.</td>
</tr>
</tbody>
</table>
Entity Property Prefixes

Lumberyard Editor supports typed properties where the type is derived from special prefixes in the property name. For a complete list of supported prefixes, refer to the `s_paramTypes` array, defined in `Objects/EntityScript.cpp`. This array maps prefixes to variable types.

The following prefixes are supported by Lumberyard:

```plaintext
{ "n", IVariable::INT, IVariable::DT_SIMPLE, SCRIPTPARAM_POSITIVE },
{ "i", IVariable::INT, IVariable::DT_SIMPLE,0 },
{ "b", IVariable::BOOL, IVariable::DT_SIMPLE,0 },
{ "f", IVariable::FLOAT, IVariable::DT_SIMPLE,0 },
{ "s", IVariable::STRING, IVariable::DT_SIMPLE,0 },
{ "ei", IVariable::INT, IVariable::DT_UIENUM,0 },
{ "es", IVariable::STRING, IVariable::DT_UIENUM,0 },
{ "shader", IVariable::STRING, IVariable::DT_SHADER,0 },
{ "clr", IVariable::VECTOR, IVariable::DT_COLOR,0 },
{ "color", IVariable::VECTOR, IVariable::DT_COLOR,0 },
{ "vector", IVariable::VECTOR, IVariable::DT_SIMPLE,0 },
{ "snd", IVariable::STRING, IVariable::DT_SOUND,0 },
{ "sound", IVariable::STRING, IVariable::DT_SOUND,0 },
{ "dialog", IVariable::STRING, IVariable::DT_DIALOG,0 },
{ "tex", IVariable::STRING, IVariable::DT_TEXTURE,0 },
{ "texture", IVariable::STRING, IVariable::DT_TEXTURE,0 },
{ "obj", IVariable::STRING, IVariable::DT_OBJECT,0 },
{ "object", IVariable::STRING, IVariable::DT_OBJECT,0 },
{ "file", IVariable::STRING, IVariable::DT_FILE,0 },
{ "aibehavior", IVariable::STRING, IVariable::DT_AI_BEHAVIOR,0 },
{ "aicharacter", IVariable::STRING, IVariable::DT_AI_CHARACTER,0 },
{ "aipropertieslist", IVariable::STRING, IVariable::DT_AI_PFPROPERTIESLIST },
{ "aiterritory", IVariable::STRING, IVariable::DT_AITERRITORY },
{ "aiwave", IVariable::STRING, IVariable::DT_AIWAVE,0 },
{ "text", IVariable::STRING, IVariable::DT_LOCAL_STRING,0 },
{ "equip", IVariable::STRING, IVariable::DT_EQUIP,0 },
{ "reverbpreset", IVariable::STRING, IVariable::DT_REVERBPRESET,0 },
{ "eaxpreset", IVariable::STRING, IVariable::DT_REVERBPRESET,0 },
{ "aianchor", IVariable::STRING, IVariable::DT_AI_ANCHOR,0 },
{ "soclass", IVariable::STRING, IVariable::DT_SOCLASS,0 },
{ "soclasses", IVariable::STRING, IVariable::DT_SOCLASSES,0 },
{ "sostate", IVariable::STRING, IVariable::DT_SOSTATE,0 },
{ "sostates", IVariable::STRING, IVariable::DT_SOSTATES,0 }
```
Creating a New Entity Class

The following example creates an entity class called Fan.

- Create a new entity definition file with the extension ".ent", for example "GameSDK\Entities\Fan.ent". This file will expose the entity to the engine.

```xml
<Entity
  Name="Fan"
  Script="Scripts/Entities/Fan.lua"
/>
```

- Create a new Lua script file, for example GameSDK\Entities\Scripts\Fan.lua. The Lua file will define the entity logic.

```lua
Fan = {
  type = "Fan",                                   -- can be useful for scripting

  -- instance member variables
  minrotspeed = 0,
  maxrotspeed = 1300,
  acceleration = 300,
  currrotspeed = 0,
  changespeed = 0,
  currangle = 0,

  -- following entries become automatically exposed to the editor and serialized (load/save)
  -- type is defined by the prefix (for more prefix types, search for s_paramTypes in /
  -- Editor\Objects\EntityScript.cpp)
  Properties = {
    bName = 0,                                -- boolean example, 0/1
    fName = 1.2,                              -- float example
    soundName = "",                           -- sound example
    fileModelName = "Objects/box.cgf",        -- file model
  },

  -- optional editor information
  Editor = {
    Model = "Editor/Objects/Particles.cgf",   -- optional 3d object that represents this
    Icon = "Clouds.bmp",                      -- optional 2d icon that represents this
    object in editor
  },

  -- optional. Called only once on loading a level.
  -- Consider calling self:OnReset(not System.IsEditor()); here
```

1038
The topics in this section describe the entity pool system, including how it is implemented, how to register a new entity class to be pooled, and how to debug it. For more information on using entity pools in Lumberyard Editor, see the Amazon Lumberyard User Guide.
This section includes the following topics:

- Entity Pool Definitions (p. 1040)
- Entity Pool Creation (p. 1042)
- Creating and Destroying Static Entities with Pools (p. 1043)
- Creating and Destroying Dynamic Entities with Pools (p. 1045)
- Serialization (p. 1046)
- Listener/Event Registration (p. 1047)
- Debugging Utilities (p. 1048)

The following processes must take place when creating an entity pool and preparing it for use. Each of these processes is described in more detail.

1. An entity pool is created by using the information in an entity pool definition.
2. An entity pool is populated with entity containers.
3. An entity pool is validated by testing the entity pool signature of one of the entity containers against the entity pool signature of each Entity class mapped to the pool.
4. All entities marked to be created through the pool have an entity pool bookmark created for them.
5. An entity pool bookmark is prepared from or returned to the entity pool, which is mapped to its Entity class on demand.

Editor Usage

When running in Lumberyard Editor, the entity pool system is not fully enabled. All entities are created outside the pools when playing in-game in the Editor. However, all flow node actions with entity pools will still work in Lumberyard Editor, mimicking the final results that you will see in-game.

**Note**

The entity pool listeners OnEntityPreparedFromPool and OnEntityReturnedToPool are still called in the Editor, even though the entity itself is not removed/reused.

Static versus Dynamic Entities

Entities can be either static or dynamic. A static entity is placed in the Editor and exported with the level. This entity always exists. A property associated with the exported information determines whether it should be pooled (and not created during level load) or instead have an entity pool bookmark made for it. A dynamic entity is created at run-time, usually from game code. The information is constructed at run-time, usually just before it is created, and passed on to the Entity system for handling. This information also indicates whether or not it should go through an entity pool.

Entity Pool Definitions

Entity pools must be defined in the file \Game\Scripts\Entities\EntityPoolDefinitions.xml. An entity pool definition is responsible for defining the following:

- the empty class that will be used by entity containers when they're not in use
- the entity classes mapped to the pool
- other properties that describe the pool and how it is used.

In general, a pool is initially filled with a defined number of entity containers; that is, empty CEntity classes with all the required entity proxies and game object extensions that are normally created when
an entity belonging to an entity class mapped to the definition is fully instantiated. For example, a normal AI entity will have the following entity proxies: sound extension, script extension, render extension, and the game object as its user extension; as its game object extension, it will have the CPlayer class. All of these classes are instantiated for each empty CEntity instance, and is reused by the entities as they are created from the pool.

The following illustrates an entity pool definition:

```xml
<Definition name="AI" emptyClass="NullAI" maxSize="16" hasAI="1" defaultBookmarked="0" forcedBookmarked="0">
  <Contains>
    <Class>Grunt</Class>
    <Class>Flyer</Class>
  </Contains>
</Definition>
```

**Empty Class**

The empty class is defined using the `emptyClass` attribute, which takes the name of a valid entity class. The purpose of the empty class is to:

- satisfy the engine's requirement to have an entity class associated with an entity at all times; an empty container is initialized/reused to this entity class
- prepare all needed entity proxies and game object extensions needed for the entities

For example, building on the definition shown in the previous section, you would create an empty class called "NullAI" and register it the same way as the other AI classes above. Then:

1. Declare the entity class and map it to its Lua script via the game factory.

```cpp
REGISTER_FACTORY(pFramework, "NullAI", CPlayer, true);
```

2. Create the Lua script for it. View sample code at \Game\Scripts\Entities\AI\NullAI.lua.

These steps will allow Lumberyard to see "NullAI" as a valid entity class. In addition, by mapping CPlayer to it, you ensure that the correct game object extension is instantiated for the entity containers. The Lua script needs to create all the entity proxies for the entity containers. In the sample code, a render proxy is created, even though we aren't loading an asset model for this entity. For more details, see the discussion of entity pool signatures in Entity Pool Creation (p. 1042).

**Entity Class Mapping**

In an entity pool definition file, the `<Contains>` section should include maps to all the entity classes that an entity must belong to when it is created through this pool. You can map as many as you want by adding a new `<Class>` node within this section. It is important that each entity have the same dynamic class hierarchy as the empty class when fully instantiated. See Debugging Utilities (p. 1048) for useful debugging tools to verify that this is the case.

**Other Properties**

An entity pool definition can define the following additional properties.
name

Unique identity given to an entity pool, useful for debugging purposes. The name should be unique across all definitions.

maxSize

Largest pool size this pool can reach. By default, this is also the number of entity containers created to fill the pool when loading a level. This value can be overwritten for a level by including an EntityPools.xml file inside the level’s root directory. This file can only be used to decrease the number of entity containers created per pool; it cannot exceed the maxSize value defined here. This is useful when you need to reduce the memory footprint of the entity pools per level. The following example file adjusts the size of an AI entity pool to “2”.

```
LevelEntityPools.xml

<EntityPools>
  <AI count="2" />
</EntityPools>
```

hasAI

Boolean value that indicates whether or not the entity pool will contain entities that have AI associated with them. It is important to set this property to TRUE if you are pooling entities with AI.

defaultBookmarked

Boolean value that indicates whether or not an entity belonging to one of the entity classes mapped to this pool is flagged as “created through pool” (see Creating and Destroying Static Entities with Pools (p. 1043)). This flag determines whether or not, during a level load, an entity pool bookmark is created for the entity instead of being instantiated.

forcedBookmarked

Boolean value that indicates whether or not an entity belonging to one of the entity classes mapped to this pool must be created through the pool. This property overrides an entity's "created through pool" flag (see Creating and Destroying Static Entities with Pools (p. 1043)).

Entity Pool Creation

When loading a level, an entity pool is created for each entity pool definition. On creation, the pool is filled with empty containers (instances of CEntity using the emptyClass attribute value as the entity class. These empty containers come with some expectations that must be satisfied:

- Containers should be minimal in size. This means you should not load any assets or large amounts of data into them. For example, in the sample Lua script (Game\Scripts\Entities\AI \NullAI.lua), the NullAI entity does not define a character model, animation graph, body damage definition, etc.
- Containers should have the same entity proxies and game object extensions created for them as compared to a CEntity fully instantiated using each of the mapped entity classes.

Once the pool is created, an entity pool signature is generated using one of the empty containers. An entity pool’s signature is a simple container that maps the dynamic class hierarchy of an entity.

One of the functions of the entity pool system is to avoid as much as possible dynamic allocation for delegate classes used by entities. Key examples of these are the entity proxies and game object extensions used by entities. When an entity pool's empty containers are first created, the delegate
classes that will be used by the real entities contained in them are also supposed to be created. To ensure that this is the case, the entity pool signature is used. It works as follows:

1. A TSerialize writer is created. It is passed to each entity proxy and game object extension that exists in the entity.
2. Each proxy and extension is expected to write some info to the TSerialize writer. This information should be unique.
3. Two signatures can then be compared to see if they contain the same written information, verifying they contain the same dynamic class hierarchy.

All of the entity proxies have already been set up to write their information to the TSerialize writer. However, if you create a new game object extension (or a new entity proxy), then you will need to set the class up to respond to the Signature helper when needed. To do this, implement the virtual method (Entity Proxy: GetSignature; Game Object Extension: GetEntityPoolSignature) and write information about the class to the TSerialize writer. Generally, all that is needed is to just begin/end a group with the class name. The function should then return TRUE to mark that the signature is valid thus far.

```cpp
CActor::GetEntityPoolSignature Example

bool CActor::GetEntityPoolSignature( TSerialize signature )
{
    signature.BeginGroup("Actor");
    signature.EndGroup();
    return true;
}
```

The section Debugging Utilities (p. 1048) discusses how to view the results of entity pool signature tests in order to verify that everything is working as expected.

Creating and Destroying Static Entities with Pools

This topic covers issues related to handling static entities.

Entity Pool Bookmarks

When an entity is marked to be created through the pool, it is not instantiated during the level load process. Instead, an entity pool bookmark is generated for it. The bookmark contains several items:

- Entity ID reserved for the entity, assigned when the level was exported. You will use this entity ID later to tell the system to create the entity.
- Static instanced data that makes the entity unique. This includes the <EntityInfo> section from the mission.xml file, which contains area information, flow graph information, child/parent links, PropertiesInstance table, etc.
-Serialized state of the entity if it has been returned to the pool in the past. See more details in Serialization (p. 1046).

In each entity's <EntityInfo> section in the mission.xml file (generated when the level is exported from the Editor), there's a CreatedThroughPool property. This property can be referenced from the SEntitySpawnParams struct. If set to TRUE, the EntityLoadManager module will not create a CEntity instance for the entity. Instead, it will delegate the static instanced data and reserved entity ID to the EntityPoolManager to create a bookmark.

```cpp
CEntityLoadManager::ParseEntities
```
Preparing a Static Entity

To prepare a static entity, call IEntityPoolManager::PrepareFromPool, passing in the entity ID associated with the static entity you want to create. In response, the following execution flow takes place:

1. System determines if the request can be processed in this frame. It will attempt to queue up multiple requests per frame and spread them out. If the parameter bPrepareNow is set to TRUE or if no prepare requests have been handled this frame, the request will be handled immediately. Otherwise, it will be added to the queue. Inside CEntityPoolManager::LoadBookmarkedFromPool, the EntityLoadManager is requested to create the entity.

   **Note**
   Note: If this activity is happening in the Editor, the entity will simply have its Enable event called. This will mimic enabling the entity via Flow Graph (unhide it). In this situation, the execution flow skips to the final step.

2. System searches for an entity container (either empty, or still in use) to hold the requested entity. The function CEntityPoolManager::GetPoolEntity looks through the active entity pools to find one that contains the entity class of the given static entity. Once the correct pool is found, the container is retrieved from it. The actual order is as follows:
   a. If a forcedPoolId (entity ID of one of the empty containers created to populate the pool) is requested, find that entity container and return it.
   b. If no forcedPoolId is requested, get an entity container from the inactive set (entity containers not currently in use).
   c. If no inactive containers are available, get one from the active set (entity containers currently in use). This action uses a "weight" value to determine which container to return. A special Lua function in the script is used to request weights for each empty container (CEntityPoolManager::GetReturnToPoolWeight). A negative weight means it should not be used at all if possible. The system might pass in an urgent flag, which means the pool is at its maximum size.
   d. If an empty container can still not be found, an urgent flag will be ignored and the system will try to grow the pool. This is only possible if the pool was not created at its maximum size (this happens when the maximum pool size is overridden for a level with a smaller maximum size). In this case, a new entity container is generated, added to the pool, and immediately used.

3. The retrieved entity container, along with the static instanced data and reserved entity ID gathered from its bookmark, is passed on through the function CEntityLoadManager::CreateEntity, which begins the Reload process. CreateEntity uses the provided entity container instead of creating a new CEntity instance. It will handle calling the Reload pipeline on the entity container, and then install all the static instanced data for the prepared static entity. The Reload pipeline is as follows:
a. The function `CEntity::ReloadEntity` is called on the entity container. The `CEntity` instance will clean itself up internally and begin using the static instanced data of the entity being prepared. The Lua script also performs cleanup using the function `OnBeingReused`.

b. The Entity system's salt buffer and other internal containers are updated to reflect that this entity container now holds the reserved entity ID and can be retrieved using it.

c. Entity proxies are prompted to reload using the static instanced data provided. This is done by calling `IEntityProxy::Reload`; each proxy is expected to correctly reset itself with the new data provided. The Script proxy is always the first to be reloaded so that the Lua script can be correctly associated before the other proxies attempt to use it.

If the game object is being used as the User proxy, all the game object extensions for the container are also prompted to reload. This is done by calling `IGameObjectExtension::ReloadExtension` on all extensions. If this function returns FALSE, the extension will be deleted. Once this is done, `IGameObjectExtension::PostReloadExtension` is called on all extensions. This behavior mimics the Init and PostInit logic. Each extension is expected to correctly reset itself with the new data provided.

4. If any serialized data exists within the bookmark, the entity container is loaded with that data. This ensures that the static entity resumes the state it was in last time it was returned to the pool. This process is skipped if this is the first time the static entity is being prepared.

At this point, calling `CEntity::GetEntity` or `CEntity::FindEntityByName` will return the entity container that is now housing the static entity and its information.

**Returning a Static Entity to the Pool**

To return a static entity, call the function `IEntityPoolManager::ReturnToPool`. You must pass in the entity ID associated with the static entity. In response, the following execution flow takes place:

1. The function `CEntityPoolManager::ReturnToPool` finds the bookmark and the entity pool containing the current entity container housing the static entity.

2. Depending on the `bSaveState` argument, the `CEntity` instance is (saved) and its serialized information is added to the bookmark. This ensures that if the static entity is prepared again later, it will resume its current state.

3. The entity container goes through the Reload process again. This time, however, the entity container is reloaded using its empty class, effectively removing all references to loaded assets/content and put it back into a minimal state.

At this point, calling `CEntity::GetEntity` or `CEntity::FindEntityByName` to find the static entity will return NULL.

**Creating and Destroying Dynamic Entities with Pools**

The processes for creating and destroying dynamic entities are similar to those for static entities, which one key exception: dynamic entities have no entity pool bookmarks (at least initially). Because they are not exported in the level, they have no static instanced data associated with them and so no bookmark is made for them.

**Creating a Dynamic Entity**

As with static entities, creating a dynamic entity with the pool starts with calling `IEntitySystem::SpawnEntity`. Construct an `SEntitySpawnParams` instance to describe its static instanced data. When filling in this struct, set the `bCreatedThroughPool` property to TRUE if you wish...
to have the entity be created through the pool. In the following example, a vehicle part from the Vehicle system is being spawned through the pool:

```cpp
SEntitySpawnParams spawnParams;
spawnParams.sName = pPartName
spawnParams.pClass = gEnv->pEntitySystem->GetClassRegistry()->FindClass("VehiclePartDetached");
spawnParams.nFlags = ENTITY_FLAG_CLIENT_ONLY;
spawnParams.bCreatedThroughPool = true;
IEntity* pSpawnedDebris = gEnv->pEntitySystem->SpawnEntity(spawnParams);
```

Once SpawnEntity, the following execution flow takes place:

1. CEntitySystem::SpawnEntity will check for an entity pool associated with the provided entity class. If so, it will delegate the workload to the entity pool manager.
2. From within CEntityPoolManager::PrepareDynamicFromPool, an entity pool bookmark is created for the new entity. This is done primarily for serialization purposes.
3. The execution flow follows the same sequence as preparing a static entity (see Creating and Destroying Static Entities with Pools (p. 1043)).
4. If the process is successful, the entity container now housing the information is returned. Otherwise, SpawnEntity creates a new CEntity instance to satisfy the request.

At this point, calling CEntity::GetEntity or CEntity::FindEntityByName will return the entity container now housing the dynamic entity and its information.

**Destroying a Dynamic Entity with the Pool**

As with static entities, use IEntitySystem::RemoveEntity or any other method that can destroy an entity. The entity pool manager will return the entity container to the pool, freeing it for use elsewhere and removing the dynamic entity in the process. The resulting execution flow differ from destroying static entities in two ways:

- Dynamic entities are not serialized when they are returned.
- The entity pool bookmark associated with the dynamic entity is removed. It is no longer needed.

At this point, calling CEntity::GetEntity or CEntity::FindEntityByName will return NULL.

**Serialization**

All entities created or prepared through the entity pool system are serialized by the system for game save/load. For this reason, do not serialize those entities marked as coming from the pool (IEntity::IsFromPool) in your normal serialization. This is handled in Lumberyard's default implementation for saving and loading the game state.

The entity pool system is serialized from the Entity system's implementation of the Serialize function.

**Saving Entity Pools**

The following process occurs when the game state is being saved:

1. All active entity containers in all entity pools are updated. This results in CEntityPoolManager::UpdatePoolBookmark being called for each active entity container. As long as the entity does not have the ENTITY_FLAG_NO_SAVE flag set on it, the bookmark is serialized as follows:
a. Serialize Helper writes to the bookmark's pLastState (an ISerializedObject), which contains
   the serialized state of the entity.

b. The callback CEntityPoolManager::OnBookmarkEntitySerialize runs through the
   serialization process on the entity. This ensures that the general information, properties and all
   entity proxies are serialized using their overloaded Serialize() implementation.

c. Any listeners subscribed to the OnBookmarkEntitySerialize callback are able to write data into
   the bookmark at this time. This is used to also bookmark AI objects along with the entity.

2. All entity pool bookmarks are saved, including the static entity and dynamic entity usage counts.

3. If any prepare requests are currently queued, the prepare request queue is saved.

## Loading Entity Pools

The following process occurs when the game state is being loaded:

1. The saved entity pool bookmarks are read in. If the bookmark is marked as containing a dynamic
   entity, it is read to ensure it exists. Each bookmark's pLastState is read in and updated.

2. If the entity pool bookmark contains an entity that was active at the time the game was saved, the
   entity is created/prepared from the pool once more.
   a. While the entity is being created/prepared, it will load its internal state using the pLastState at
      its final step, because the object contains information at this point.
   b. This will also call the OnBookmarkEntitySerialize listener callback, allowing other systems to
      read data from the bookmark.

## Listener/Event Registration

There are several listener and various event callbacks dealing with entity pool usage. These callbacks
are important for sub-systems that rely on entity registration. They can notify you when an entity has
been prepared or returned to the pool so that you can register and unregister it with your subsystems as
needed.

### IEntityPoolListener

This listener can be subscribed to via IEntityPoolManager::AddListener. It contains the following
callbacks:

- **OnPoolBookmarkCreated**
  Called when an entity pool bookmark has been created. The reserved entity ID for the pooled entity
  is passed in, along with the static instanced data belonging to it.

- **OnEntityPreparedFromPool**
  Called when an entity (static or dynamic) has been prepared from the pool. You are given both the
  entity ID and the entity container that is now housing the entity. This is called at the end of the
  prepare entity process.

- **OnEntityReturnedToPool**
  Called when an entity (static or dynamic) has been returned to the pool. You are given both the
  entity ID and the entity container that is currently housing the entity. This is called at the start of the
  return entity process.

- **OnPoolDefinitionsLoaded**
  Called at initialization, with information allowing listeners to set up their own resources for working
  with the pool. Currently passes the total number of pooled entities that have AI attached.
OnBookmarkEntitySerialize

Called during reads and writes from entity bookmarks, allowing listeners to store additional data in
the bookmark.

IEntitySystemSink

This listener has a special callback, OnReused, that notifies you when an entity has been reloaded. This is
the process an entity container goes through when a static entity is being prepared into it, or a dynamic
entity is being created inside it. You are given the entity container that houses the entity as well as the
static instanced data belonging to it.

Debugging Utilities

There are several debugging utilities you can use to manage the entity pools and see how they are being
used during gameplay.

Debugging Entity Pool Bookmarks

To see the status of all entity pool bookmarks that currently exist during the game, use the following
console command.

```
es_dump_bookmarks [filterName] [dumpToDisk]
```

This command causes text to be written to the console and game log file for every bookmark requested.

Arguments

**filterName**

(Optional) Allows you to filter your request to get bookmarks only for entities whose names contain
the specified value as a substring. To display all bookmarks, set this argument to "all" or leave it empty.

**dumpToDisk**

(Optional) Allows you to output to disk all static instanced data associated with the displayed
bookmarks. If supplied and its a non-zero numerical file, data will be stored at \User\Bookmarks
\LevelName\EntityName.xml.

Data displayed

The following information is displayed for each bookmark:

- Name of the bookmarked entity.
- Layer the bookmarked entity belongs to.
- Entity class name the bookmarked entity uses.
- Reserved entity ID associated with the bookmarked entity.
- If the bookmarked entity has the No Save Entity Flag associated with it.
- If the bookmarked entity is static or dynamic.
- If the bookmarked entity contains any serialized data (and the memory footprint of the information if
  available).
- If the bookmarked entity contains any static instanced data (and the memory footprint of the
  information if available).
Entity ID Explained

When referring to a dynamic C++ object, pointers and reference counting can be used, but a better method is to use a weak reference that allows you to remove an object and have all references become invalid. This option limits the need to iterate over all objects to invalidate objects being removed, which results in performance costs.

With each reference, Lumberyard stores a number called the "salt" (also called a "magic number"). This number, together with the index, gives the object a unique entity ID over the game lifetime. Whenever an object is destroyed and the index is reused, the salt is increased and all references with the same index become invalid. To get an entity position/pointer, the entity manager needs to resolve the entity ID; as the salt is different, the method fails.

The class CSaltBufferArray handles adding and removing objects and does the required adjustments to the salt. The object array is kept compact for more cache-friendly memory access. Storing EntityId references to disc is possible and used for saved games and by the Editor game export. However, when loading a saved game of a level that has been patched and now has more entities, this can result in a severe conflict. To solve this problem, dynamic entities are created starting with a high index counting down, while static entities are created starting with a low index counting up.

Entity IDs have the following limitations:

- A 16-bit index allows up to approximately 65,000 living objects. This should be enough for any non-massive multiplayer game. In a massive multiplayer game, the method described here should not be used by the server. However, it can be used between specific clients and the server.
- A 16-bit salt value allows a game to reuse an index up to approximately 65,000 times. If that happens, the index can no longer be used. This should be enough for any non-massive multiplayer game, when used with some care—don't create and destroy objects (such as bullets) too rapidly. A massive multiplayer game, or any game that supports multi-day game sessions, can run into this limit.

Adding Usable Support on an Entity

Overview

Players may be able to interact with an entity using a key press ('F' by default). Entities that can be interacted with will be enabled with a special on-screen icon inside the game to inform the player that interaction is possible.

To use this feature, you need to create a script that implements two functions: IsUsable() and OnUsed().

Preparing the Script

The script should look like this:

```
MakeUsable(NewEntity)

function NewEntity:IsUsable(userId)
    -- code implementation
    return index;
end

function NewEntity:OnUsed(userId, index)
    -- code implementation
end
```
Implementing IsUsable

The IsUsable() function is called when a player is aiming the cross-hairs towards the entity. The function will determine if the entity can be interacted with by the player doing the aiming. The function only accepts a single parameter: the player's entity ID.

If the player cannot interact with the entity, the function should return 0. This value causes the UI to not render the "USE" icon over the entity.

If the player can interact with the entity, the function should return a positive value. This value will be stored and later used when calling the OnUsed() function.

Implementing OnUsed

The OnUsed() function is called when a player presses interacts with the entity (such as by pressing the Use key when the USE icon is active. This function accepts two parameters: (1) the player's entity ID, and (2) the value returned by IsUsable().

Entity Scripting

This section contains topics on using Lua scripting to work with the legacy Entity system. These Lua scripts use the legacy script context.

Starting with Lumberyard 1.8, Lua scripts use the new behavior context that replaces the legacy script context. Scripts that were written before the integration of the behavior context no longer work in Lumberyard versions 1.8 and later. For information on updating Lua code from legacy script context to the new behavior context, see the migration notes for Lumberyard 1.8.

For information on using Lua with Lumberyard's new component entity system, see Writing Lua Scripts for the Component Entity System.

This section includes the following topics:

- Structure of a Script Entity (p. 1050)
- Using Entity State (p. 1054)
- Using Entity Slots (p. 1055)
- Linking Entities (p. 1057)
- Exposing an Entity to the Network (p. 1058)

Structure of a Script Entity

This page contains information on using Lua scripting to work with the legacy Entity system. These Lua scripts use the legacy script context. Starting with Lumberyard 1.8, Lua scripts use the new behavior context that replaces the legacy script context.
Scripts that were written before the integration of the behavior context no longer work in Lumberyard versions 1.8 and later. For information on updating Lua code from legacy script context to the new behavior context, see the migration notes for Lumberyard 1.8. For information on using Lua with Lumberyard's new component entity system, see Writing Lua Scripts for the Component Entity System.

To implement a new entity using Lua, two files need to be created and stored in the game directory:

- The Ent file tells the Entity system the location of the Lua script file.
- The Lua script file implements the desired properties and functions.

With the SDK, both the .ent and .lua files are stored inside the `<Game_Folder>\Scripts.pak` file.

**Ent File**

The Ent files are all stored inside the `<Game_Folder>\Entities` directory and need to have the .ent file extension. The content is XML as follows:

```xml
<Entity
    Name="LivingEntity"
    Script="Scripts/Entities/Physics/LivingEntity.lua"
/>
```

Entity properties set in the Ent file include:

**Name**

Name of the entity class.

**Script**

Path to the Lua script that implements the entity class.

**Invisible**

Flag indicating whether or not the entity class is visible in Lumberyard Editor.

**Lua Script**

The Lua script, in addition to implementing the entity class, provides a set of information used by Lumberyard Editor when working with entities on a level. The property values set inside the Lua script are default values assigned to new entity instances. Editor variables specify how entities are drawn in Lumberyard Editor.

The following code excerpt is from the sample project files in your Lumberyard directory (`\dev\Cache\SamplesProject\pc\samplesproject\scripts\entities\physics\livingentity.lua`).

```lua
LivingEntity = {
    Properties = {
        soclasses_SmartObjectClass = "",
        bMissionCritical = 0,
        bCanTriggerAreas = 1,
        DmgFactorWhenCollidingAI = 1,
        object_Model = "objects/default/primitive_capsule.cgf",
    }
    Physics = {
        bPhysicalize = 1, -- True if object should be physicalized at all.
        bPushableByPlayers = 1,
    }
}
```
Lumberyard Legacy Reference
Structure of a Script Entity

Living = {
    height = 0, -- vertical offset of collision geometry center
    vector_size = {0.4, 0.4, 0.9}, -- collision cylinder dimensions
    height_eye = 1.8, -- vertical offset of camera
    height_pivot = 0.1, -- offset from central ground position that is considered entity center
    head_radius = 0.3, -- radius of the 'head' geometry (used for camera offset)
    head_height = 1.7, -- center.z of the head geometry
    groundContactEps = 0.004, -- the amount that the living needs to move upwards before ground contact is lost. Defaults to which ever is greater 0.004, or 0.01*geometryHeight
    bUseCapsule = 1, -- switches between capsule and cylinder collider geometry
    inertia = 1, -- inertia koefficient, the more it is, the less inertia is, 0 means no inertia
    inertiaAccel = 1, -- inertia on acceleration
    air_control = 1, -- air control koefficient 0..1, 1 - special value (total control of movement)
    air_resistance = 0.1, -- standard air resistance
    gravity = 9.8, -- gravity vector
    mass = 100, -- mass (in kg)
    min_slide_angle = 60, -- if surface slope is more than this angle, player starts sliding (angle is in radians)
    max_climb_angle = 60, -- player cannot climb surface which slope is steeper than this angle
    max_jump_angle = 45, -- player is not allowed to jump towards ground if this angle is exceeded
    min_fall_angle = 65, -- player starts falling when slope is steeper than this
    max_vel_ground = 10, -- player cannot stand of surfaces that are moving faster than this
    timeImpulseRecover = 0.3, -- forcefully turns on inertia for that duration after receiving an impulse
    nod_speed = 1, -- vertical camera shake speed after landings
    bActive = 1, -- 0 disables all simulation for the character, apart from moving along the requested velocity
    collision_types = 271, -- (271 = ent_static | ent_terrain | ent_living | ent_rigid | ent_sleeping_rigid) entity types to check collisions against

    MultiplayerOptions = {
        bNetworked = 0,
    },

    bExcludeCover = 0,
},

Client = {},
Server = {},
-- Temp.
_FLAGS = {},

Editor={
    Icon = "physicsobject.bmp",
    IconOnTop = 1,
},

This information is followed by functions that implement the entity class.

Properties

Entity properties are placed inside the entity class. These properties are assigned to all new instances of the entity class created, visible and editable in Lumberyard Editor as the instance's Entity Properties.
table. The property values set for individual entity instances placed on a level are saved in the level file. When a property of an entity instance is changed in Lumberyard Editor, the `OnPropertyChanged()` function called (if it is has been implemented for the script entity).

Lumberyard Editor provides the Archetype tool for assigning a common set of properties reused for multiple instance (even across multiple levels). For more information on Archetypes, see Archetype Entity in the Amazon Lumberyard User Guide.

When specifying entity class property names, use the following prefixes to signal the data type expected for a property value. This enables Lumberyard Editor to validate a property value when set.

### Entity class property prefixes

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>boolean</td>
</tr>
<tr>
<td>f</td>
<td>float</td>
</tr>
<tr>
<td>i</td>
<td>integer</td>
</tr>
<tr>
<td>n</td>
<td>positive integer</td>
</tr>
<tr>
<td>s</td>
<td>string</td>
</tr>
<tr>
<td>clr</td>
<td>color</td>
</tr>
<tr>
<td>object_</td>
<td>object compatible with Lumberyard (CFG, CGA, CHR or CDF file)</td>
</tr>
</tbody>
</table>

You can add special comments to property values that can be utilized by the engine. For example:

```plaintext
--[25,100,0.1,"Damage threshold"]
```

This comment tells the engine the following:

- Value is limited to between 25 and 100.
- The float value uses a step of 0.01 (this limits the fidelity of values).
- The string "Damage threshold" will be displayed in Lumberyard Editor as a tool tip.

### Editor Table

The Editor table provides additional configuration information to Lumberyard Editor on how to handle instances of the entity.

### Entity class editor variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>CGF model to be rendered over an entity instance.</td>
</tr>
<tr>
<td>ShowBounds</td>
<td>Flag indicating whether or not a bounding box is drawn around an entity instance when selected.</td>
</tr>
<tr>
<td>AbsoluteRadius</td>
<td></td>
</tr>
<tr>
<td>Icon</td>
<td>BMP icon to be drawn over an entity instance.</td>
</tr>
</tbody>
</table>
### Variable Description

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IconOnTop</td>
<td>Flag indicating whether or not the icon is drawn over or under an entity instance.</td>
</tr>
<tr>
<td>DisplayArrow</td>
<td></td>
</tr>
<tr>
<td>Links</td>
<td></td>
</tr>
</tbody>
</table>

### Functions

A script entity can include several callback functions called by the engine or game system. See Entity System Script Callbacks (p. 735) for more information.

### Using Entity State

CryLua is deprecated and will be removed in a future version of Lumberyard.

**Note**
This page contains information on using Lua scripting to work with the legacy Entity system. These Lua scripts use the legacy script context. Starting with Lumberyard 1.8, Lua scripts use the new behavior context that replaces the legacy script context. Scripts that were written before the integration of the behavior context no longer work in Lumberyard versions 1.8 and later. For information on updating Lua code from legacy script context to the new behavior context, see the migration notes for Lumberyard 1.8. For information on using Lua with Lumberyard's new component entity system, see Writing Lua Scripts for the Component Entity System.

The Entity system provides a simple state-switching mechanism for script entities.

Each state consists of the following:

- Name (string)
- Lua table within the entity table, identified with the state name
- OnEndState() function (optional)
- OnBeginState() function (optional)
- Additional callback functions (optional) (See Entity System Script Callbacks (p. 735))

**To declare states for an entity:**

All entity states must be declared in the entity's main table to make the Entity system aware of them. The following examples show how to declare "Opened", "Closed", and "Destroyed" states.

```lua
AdvancedDoor =
{
    Client = {},
    Server = {},
    PropertiesInstance = ...,
    Properties = ...,
    States = {"Opened","Closed","Destroyed"},
}
```
**To define an entity state:**

Entity states can be either on the server or client (or both). The definition for a server-side "Opened" state might look as follows:

```lua
AdvancedDoor.Server.Opened = 
{
    OnBeginState = function( self )
        if(self.Properties.bUsePortal==1)then
            System.ActivatePortal(self:GetWorldPos(),1,self.id);
        end;
        self.bUpdate=1;
        self.lasttime=0;
        AI.ModifySmartObjectStates( self.id, "Open-Closed" );
        self:Play(1);
        end,

    OnUpdate = function(self, dt)
        self:OnUpdate();
        end,
}
```

**To set an entity's initial state:**

Initially, an entity has no state. To set an entity's state, use one of the entity's callback functions (not to be confused with an entity state's callback function) to call its `GotoState()` method, shown in the following example. Once the entity state is set, the entity resides in that state and events will also be directed to that state.

```lua
function AdvancedDoor:OnReset()
    self:GotoState("Opened");
end
```

**To change an entity's state:**

Transitioning from the current state to any other state can also be done using the `GotoState()` method, as follows.

```lua
function AdvancedDoor.Server:OnHit(hit)
    ...
    if(self:IsDead())then
        self:GotoState("Destroyed");
    end
end
```

**To query an entity's state:**

Querying the state the entity is currently in can be done using the `GetState()` method, as follows.

```lua
if (self:GetState()=="Opened") then ...
if (self:GetState()~="Opened") then ...
```

---

**Using Entity Slots**

CryLua is deprecated and will be removed in a future version of Lumberyard.
Note
This page contains information on using Lua scripting to work with the legacy Entity system. These Lua scripts use the legacy script context. Starting with Lumberyard 1.8, Lua scripts use the new behavior context that replaces the legacy script context. Scripts that were written before the integration of the behavior context no longer work in Lumberyard versions 1.8 and later. For information on updating Lua code from legacy script context to the new behavior context, see the migration notes for Lumberyard 1.8. For information on using Lua with Lumberyard’s new component entity system, see Writing Lua Scripts for the Component Entity System.

Each entity can have slots that are used to hold different resources available in Lumberyard. This topic describes how to work with entity slots.

**Allocating a Slot**

The following table lists the resources that can be allocated in a slot, along with the ScriptBind function used to allocate it.

<table>
<thead>
<tr>
<th>Lumberyard resource</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>static geometry</td>
<td>LoadObject() or LoadSubObject()</td>
</tr>
<tr>
<td>animated character</td>
<td>LoadCharacter()</td>
</tr>
<tr>
<td>particle emitter</td>
<td>LoadParticleEffect()</td>
</tr>
<tr>
<td>light</td>
<td>LoadLight()</td>
</tr>
<tr>
<td>cloud</td>
<td>LoadCloud()</td>
</tr>
<tr>
<td>fog</td>
<td>LoadFogVolume()</td>
</tr>
<tr>
<td>volume</td>
<td>LoadVolumeObject()</td>
</tr>
</tbody>
</table>

**Modifying Slot Parameters**

Each of these resource may be moved, rotated, or scaled relative to the entity itself.

- SetSlotPos()
- GetSlotPos()
- SetSlotAngles()
- GetSlotAngles()
- SetSlotScale()
- GetSlotScale()

You can add a parenting link between the slots, making it possible to have related positions.

- SetParentSlot()
- GetParentSlot()

**Slot Management**

To determine whether or not a specified slot is allocated, call the function IsSlotValid().
To free one slot, call `!FreeSlot()`.

To free all allocated slots within the entity, call `!FreeAllSlots()`.

**Loading a Slot**

The following example illustrates loading a slot in a script function.

```lua
local pos={x=0,y=0,z=0};
self:LoadObject(0,props.fileModel);
self:SetSlotPos(0,pos);
self:SetCurrentSlot(0);
```

**Linking Entities**

CryLua is deprecated and will be removed in a future version of Lumberyard.

**Note**

This page contains information on using Lua scripting to work with the legacy Entity system. These Lua scripts use the legacy script context. Starting with Lumberyard 1.8, Lua scripts use the new behavior context that replaces the legacy script context. Scripts that were written before the integration of the behavior context no longer work in Lumberyard versions 1.8 and later. For information on updating Lua code from legacy script context to the new behavior context, see the migration notes for Lumberyard 1.8. For information on using Lua with Lumberyard's new component entity system, see Writing Lua Scripts for the Component Entity System.

In Lumberyard Editor, you can link an entity to other entities. These links are organized inside the Entity system. Each entity can link to multiple entities. Each link has a name associated to it. See the Amazon Lumberyard User Guide for more information about grouping and linking objects.

The following example Lua script searches the Entity system for any links to other entities that are named "Generator".

```lua
function RadarBase:IsPowered()
local i=0;
local link = self:GetLinkTarget("Generator", i);

while (link) do
  Log("Generator %s", link:GetName());
  if (link:GetState() == "PowerOn") then
    if (link.PowerConnect) then
      link:PowerConnect(self.id);
      return true;
    end
  end
  i=i+1;
  link=self:GetLinkTarget("Generator", i);
end

return false;
end
```
The legacy ScriptBind functions related to entity links are listed in Lua ScriptBind Reference (p. 757). The following functions are used to read or create entity links:

- CountLinks
- CreateLink
- GetLink
- GetLinkName
- GetLinkTarget
- RemoveAllLinks
- RemoveLink
- SetLinkTarget

Exposing an Entity to the Network

CryLua is deprecated and will be removed in a future version of Lumberyard.

**Note**
This page contains information on using Lua scripting to work with the legacy Entity system. These Lua scripts use the legacy script context. Starting with Lumberyard 1.8, Lua scripts use the new behavior context that replaces the legacy script context. Scripts that were written before the integration of the behavior context no longer work in Lumberyard versions 1.8 and later. For information on updating Lua code from legacy script context to the new behavior context, see the migration notes for Lumberyard 1.8. For information on using Lua with Lumberyard’s new component entity system, see Writing Lua Scripts for the Component Entity System and the Network Binding.

A script entity can be a serialized value on the network. This approach is done by setting the values on the server and having them automatically synchronized on all the clients. It also makes it possible to invoke client/server RMI functions.

Keep in mind the following limitations:

- There is no notification when a serialized value has changed.
- Values are controlled on the server only, there is no way to set values on the client.

Exposing a Script Entity to CryNetwork

To define the network features of an entity, call the ScriptBind function `Net.Expose()`, as illustrated in the following code. This code is written inside a Lua script within the global space, rather than in a function.

```lua
Net.Expose {
  Class = DeathMatch,
  ClientMethods = {
    ClVictory = { RELIABLE_ORDERED, POST_ATTACH, ENTITYID, },
    ClNoWinner = { RELIABLE_ORDERED, POST_ATTACH, },
    ClClientConnect = { RELIABLE_UNORDERED, POST_ATTACH, STRING, BOOL },
    ClClientDisconnect = { RELIABLE_UNORDERED, POST_ATTACH, STRING, },
    ClClientEnteredGame = { RELIABLE_UNORDERED, POST_ATTACH, STRING, },
  }
}```
### RMI functions

The RMI function is defined in either the ClientMethods and ServerMethods tables passed to the `Net.Expose()` function.

**Order flags:**
- `UNRELIABLE_ORDERED`
- `RELIABLE_ORDERED`
- `RELIABLE_UNORDERED`

The following descriptors control how the RMI is scheduled within the data serialization.

<table>
<thead>
<tr>
<th>RMI attach flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_ATTACH</td>
<td>No special control (preferred)</td>
</tr>
<tr>
<td>PRE_ATTACH</td>
<td>Call occurs before data serialized</td>
</tr>
<tr>
<td>POST_ATTACH</td>
<td>Call occurs after the data serialized</td>
</tr>
</tbody>
</table>

The following example shows a function declaration:

```plaintext
function DeathMatch.Client:ClClientConnect(name, reconnect)
```

The following examples illustrate a function call:

```plaintext
self.allClients:ClVictory( winningPlayerId );
```

```plaintext
self.otherClients:ClClientConnect( channelId, player:GetName(), reconnect );
```

```plaintext
self.onClient:ClClientConnect( channelId, player:GetName(), reconnect );
```

See [RMI Functions (p. 205)](#) for more details.

**Note**

Note: Script networking doesn't have an equivalent to the dependent object RMIs.

### ServerProperties table

The entity table also contains a ServerProperties table that indicates which properties need to be synchronized. This is also the place to define the variable type of the value.
Exposing a Script Entity to CryAction

In addition, you must create a game object in CryAction and bind the new game object to the network game session. The following example shows the code placed in the `OnSpawn()` function:

```
CryAction.CreateGameObjectForEntity(self.id);
CryAction.BindGameObjectToNetwork(self.id);
```

You can also instruct the game object to receive a per-frame update callback, as in the following function call to CryAction:

```
CryAction.ForceGameObjectUpdate(self.id, true);
```

The script entity receive the `OnUpdate()` function callback of its Server table.

```
function Door.Server:OnUpdate(frameTime)
    -- some code
end
```

**Note**

Adding update callback code to your script entity can decrease the performance of a game.
Memory Handling

This article discusses some memory and storage considerations related to game development.

Hardware Memory Limitations

Developing for game consoles can be challenging due to memory limitations. From a production point of view, it is tempting to use less powerful hardware for consoles, but the expectations for console quality are usually higher in an increasingly competitive market.

Choosing an Operating System or Device to Target

It is often better to choose only one development operating system or device, even if multiple operating systems or devices are targeted for production. Choosing an environment with lower memory requirements eases production in the long run, but it can degrade the quality on other devices. Some global code adjustments (for example, TIF setting "globalreduce", TIF preset setting "don't use highest LOD") can help in reducing memory usage, but often more asset-specific adjustments are needed, like using the TIF "reduce" setting. If those adjustments are insufficient, completely different assets are required (for example, all LODs of some object are different for console and PC). This can be done through a CryPak feature. It is possible to bind multiple pak files to a path and have them behave as layered. This way it is possible to customize some operating systems or devices to use different assets. Environments that use multiple layers have more overhead (memory, performance, I/O), so it is better to use multiple layers on more powerful hardware.

Budgets

Budgets are mostly game specific because all kinds of memory (for example, video/system/disk) are shared across multiple assets, and each game utilizes memory differently. It's a wise decision to dedicate a certain amount of memory to similar types of assets. For example, if all weapons roughly cost the same amount of memory, the cost of a defined number of weapons is predictable, and with some careful planning in production, late and problematic cuts can be avoided.

Allocation Strategy with Multiple Modules and Threads

The Lumberyard memory manager tries to minimize fragmentation by grouping small allocations of similar size. This is done in order to save memory, allow fast allocations and deallocations and to minimize conflicts between multiple threads (synchronization primitives for each bucket). Bigger
allocations run through the OS as that is quite efficient. It is possible to allocate memory in other than
the main thread, but this can negatively impact the readability of the code. Memory allocated in one
module should be deallocated in the same module. Violating this rule might work in some cases, but this
breaks per module allocation statistics. The simple `Release()` method ensures objects are freed in the
same module. The string class (`CryString`) has this behavior built in, which means the programmer
doesn’t need to decide where the memory should be released.

### Caching Computational Data

In general, it is better to perform skinning (vertex transformation based on joints) of characters on the
GPU. The GPU is generally faster in doing the required computations than the CPU. Caching the skinned
result is still possible, but memory is often limited on graphics hardware, which tends to be stronger on
calculations. Under these conditions, it makes sense to recompute the data for every pass, eliminating
the need to manage cache memory. This approach is advantageous because character counts can vary
significantly in dynamic game scenes.

### Compression

There are many lossy and lossless compression techniques that work efficiently for a certain kind of data.
They differ in complexity, compression and decompression time and can be asymmetric. Compression
can introduce more latency, and only few techniques can deal with broken data such as packet loss and
bit-flips.

### Disk Size

Installing modern games on a PC can be quite time consuming. Avoiding installation by running the
game directly from a DVD is a tempting choice, but DVD performance is much worse than hard drive
performance, especially for random access patterns. Consoles have restrictions on game startup times
and often require a game to cope with a limited amount of disk memory, or no disk memory at all. If a
game is too big to fit into memory, streaming is required.

### Total Size

To keep the total size of a build small, the asset count and the asset quality should be reasonable. For
production it can make sense to create all textures in double resolution and downsample the content
with the Resource Compiler. This can be useful for development on multiple operating systems and
devices and allows later release of the content with higher quality. It also eases the workflow for artists
as they often create the assets in higher resolutions anyway. Having the content available at higher
resolutions also enables the engine to render cut-scenes with the highest quality if needed (for example,
when creating videos).

Many media have a format that maximizes space, but using the larger format can cost more than using
a smaller one (for example, using another layer on a DVD). Redundancy might be a good solution to
minimize seek times (for example, storing all assets of the same level in one block).

### Address Space

Some operating systems (OSes) are still 32-bit, which means that an address in main memory has 32-
bits, which results in 4 GB of addressable memory. Unfortunately, to allow relative addressing, the top
bit is lost, which leaves only 2 GB for the application. Some OSes can be instructed to drop this limitation
by compiling applications with large address awareness, which frees up more memory. However, the
full 4 GB cannot be used because the OS also maps things like GPU memory into the memory space.
When managing that memory, another challenge appears. Even if a total of 1 GB of memory is free,
a contiguous block of 200 MB may not be available in the virtual address space. In order to avoid this
problem, memory should be managed carefully. Good practices are:
• Prefer memory from the stack with constant size (SPU stack size is small).
• Allocating from the stack with dynamic size by using `alloca()` is possible (even on SPU), but it can introduce bugs that can be hard to find.
• Allocate small objects in bigger chunks (flyweight design pattern).
• Avoid reallocations (for example, reserve and stick to maximum budgets).
• Avoid allocations during the frame (sometimes simple parameter passing can cause allocations).
• Ensure that after processing one level the memory is not fragmented more than necessary (test case: loading multiple levels one after another).

A 64-bit address space is a good solution for the problem. This requires a 64-bit OS and running the 64-bit version of the application. Running a 32-bit application on a 64-bit OS helps very little. Note that compiling for 64-bit can result in a bigger executable file size, which can in some cases be counterproductive.

**Bandwidth**

To reduce memory bandwidth usage, make use of caches, use a local memory access pattern, keep the right data nearby, or use smaller data structures. Another option is to avoid memory accesses all together by recomputing on demand instead of storing data and reading it later.

**Latency**

Different types of memory have different access performance characteristics. Careful planning of data storage location can help to improve performance. For example, blending animation for run animation needs to be accessible within a fraction of a frame, and must be accessible in memory. In contrast, cut-scene animations can be stored on disk. To overcome higher latencies, extra coding may be required. In some cases the benefit may not be worth the effort.

**Alignment**

Some CPUs require proper alignment for data access (for example, reading a float requires an address divisible by 4). Other CPUs perform slower when data is not aligned properly (misaligned data access). As caches operate on increasing sizes, there are benefits to aligning data to the new sizes. When new features are created, these structure sizes must be taken into consideration. Otherwise, the feature might not perform well or not even work.

**Virtual Memory**

Most operating systems try to handle memory quite conservatively because they never know what memory requests will come next. Code or data that has not been used for a certain time can be paged out to the hard drive. In games, this paging can result in stalls that can occur randomly, so most consoles avoid swapping.

**Streaming**

Streaming enables a game to simulate a world that is larger than limited available memory would normally allow. A secondary (usually slower) storage medium is required, and the limited resource is used as a cache. This is possible because the set of assets tends to change slowly and only part of the content is required at any given time. The set of assets kept in memory must adhere to the limits of the hardware available. While memory usage can partly be determined by code, designer decisions regarding the placement, use, and reuse of assets, and the use of occlusion and streaming hints are also important in determining the amount of memory required. Latency of streaming can be an issue when large changes to
the set of required assets are necessary. Seek times are faster on hard drives than on most other storage media like DVDs, Blue-Rays or CDs. Sorting assets and keeping redundant copies of assets can help to improve performance.

Split screen or general multi-camera support add further challenges for the streaming system. Tracking the required asset set becomes more difficult under these circumstances. Seek performance can get get worse as multiple sets now need to be supported by the same hardware. It is wise to limit gameplay so that the streaming system can perform well. A streaming system works best if it knows about the assets that will be needed beforehand. Game code that loads assets on demand without registering them first will not be capable of doing this. It is better to wrap all asset access with a handle and allow registration and creation of handles only during some startup phase. This makes it easier to create stripped down builds (minimal builds consisting only of required assets).

Streaming System

The Lumberyard streaming engine takes care of the streaming of meshes, textures, music, sounds, and animations.

Low-level Streaming System

CryCommon Interfaces and Structs

The file IStreamEngine.h in CryCommon contains all the important interfaces and structs used by the rest of the engine.

First of all there is the IStreamEngine itself. There is only one IStreamingEngine in the application and it controls all the possible I/O streams. Most of the following information comes directly from the documentation inside the code, so it's always good to read the actual code in IStreamEngine.h file for any missing information.

The most important function in IStreamEngine is the StartRead function which is used to start any streaming request.

IStreamEngine.h

```cpp
UNIQUE_IFACE struct IStreamEngine
{
public:

    // Description:
    //   Starts asynchronous read from the specified file (the file may be on a
    // virtual file system, in pak or zip file or wherever).
    //   Reads the file contents into the given buffer, up to the given size.
    //   Upon success, calls success callback. If the file is truncated or for other
    // reason can not be read, calls error callback. The callback can be NULL
    // (in this case, the client should poll the returned IReadStream object;
    // the returned object must be locked for that)
    // NOTE: the error/success/progress callbacks can also be called from INSIDE
    // this function
    // Return Value:
    //   IReadStream is reference-counted and will be automatically deleted if
    // you don't refer to it; if you don't store it immediately in an auto-pointer,
    // it may be deleted as soon as on the next line of code,
    // because the read operation may complete immediately inside StartRead()
    // and the object is self-disposed as soon as the callback is called.
    virtual IReadStreamPtr StartRead (const EStreamTaskType tSource, const char* szFile,
    IStreamCallback* pCallback = NULL, StreamReadParams* pParams = NULL) = 0;
}
```
The following are the currently supported streaming task types. This enum should be extended if you want to stream a new object type.

*IStreamEngine.h*

```c
enum EStreamTaskType
{
    eStreamTaskTypeCount      = 13,
    eStreamTaskTypePak        = 12, // Pak file itself
    eStreamTaskTypeFlash      = 11, // Flash file object
    eStreamTaskTypeVideo      = 10, // Video data (when streamed)
    eStreamTaskTypeReadAhead  = 9,  // Read ahead data used for file reading prediction
    eStreamTaskTypeShader     = 8,  // Shader combination data
    eStreamTaskTypeSound      = 7,  // Sound data
    eStreamTaskTypeMusic      = 6,  // Music data
    eStreamTaskTypeFSBCache   = 5,  // Complete FSB file
    eStreamTaskTypeAnimation  = 4,  // All the possible animations types (dba, caf, ..)
    eStreamTaskTypeTerrain    = 3,  // Partial terrain data
    eStreamTaskTypeGeometry   = 2,  // Mesh or mesh lods
    eStreamTaskTypeTexture    = 1,  // Texture mip maps (currently mip0 is not streamed)
};
```

A callback object can be provided to the StartStream function to be informed when the streaming request has finished. The callback object should implement the following `StreamAsyncOnComplete` and `StreamOnComplete` functions.

*IStreamEngine.h*

```c
class IStreamCallback
{
    public:
        // Description:
        // Signals that reading the requested data has completed (with or without error).
        // This callback is always called, whether an error occurs or not, and is called
        // from the async callback thread of the streaming engine, which happens
        // directly after the reading operation
        virtual void StreamAsyncOnComplete (IReadStream* pStream, unsigned nError) {} 

        // Description:
        // Same as the StreamAsyncOnComplete, but this function is called from the main
        // thread and is always called after the StreamAsyncOnComplete function.
        virtual void StreamOnComplete (IReadStream* pStream, unsigned nError) = 0;
};
```

When starting a read request, you can also provide the optional parameters shown in the following code.

*IStreamEngine.h*

```c
struct StreamReadParams
{
    public:
        // The user data that'll be used to call the callback.
        DWORD_PTR dwUserData;

        // The priority of this read
        EStreamTaskPriority ePriority;

        // Description:
    
};
```
The return value of the `StartRead` function is an `IReadStream` object which can be optionally stored on the client. The `IReadStream` object is refcounted internally. When the callback object can be destroyed before the reading operation is finished, the readstream should be stored separately, and the abort should be called on it. Doing this will clean up the entire read request internally and will also call the async and sync callback functions.

The `Wait` function can be used to perform a blocking reading requests on the streaming engine. This function can be used from an async reading thread that uses the Lumberyard streaming system to perform the actual reading.

`IStreamEngine.h`

```cpp
class IReadStream : public CMultiThreadRefCount
{
public:
    virtual void Abort() = 0;
    virtual void Wait( int nMaxWaitMillis=-1 ) = 0;
};
```

**Internal Flow of a Read Request**

The Lumberyard stream engine uses extra worker and IO threads internally. For every possible IO input, a different `StreamingIOThread` is created which can run independently from the others.

Currently the stream engine has the following IO threads:

- **Optical** – Streaming from the optical data drive.
- **Hard disk drive (HDD)** – Streaming from installed data on the hard disk drive (this could be a fully installed game, or shadow copied data).
- **Memory** – Streaming from packed in-memory files, which requires very little IO.
When a reading request is made on the streaming engine, it first checks which IO thread to use, and computes the sortkey. The request is then inserted into one of the StreamingIOThread objects.

After the reading operation is finished, the request is forwarded to one of the decompression threads if the data was compressed, and then into one of the async callback threads. The amount of async callback threads is dependent on the operating system, and some async callback threads are reserved for specific streaming request types such as geometry and textures. After the async callback has been processed, the finished streaming request is added to the streaming engine to be processed on the main thread. The next update on the streaming engine from the main thread will then call the sync callback (StreamOnComplete) and clean up the temporary allocated memory if needed.

For information regarding the IO/WorkerThreads please check the StreamingIOThread and StreamingWorkerThread class.

Read Request Sorting

Requests to the streaming engine are not processed in a the same order as which they have been requested. The system tries to internally ‘optimize’ the order in which to read the data, to maximize the read bandwidth.

When reading data from an optical disc, it is very important to reduce the amount of seeks. (This is also true when reading from a hard disk drive, but to a lesser extent). A single seek can take over 100 milliseconds, while the actual read time might take only a few milliseconds. Some official statistics from the 360 XDK follow.

- Outer diameter throughput : 12x (approximately 15 MB per second).
- Inner diameter throughput : 5x (6.8 MB per second).
- Average seek (1/3rd stroke) time : 110 ms typical, 140 ms maximum.
- Full stroke seek time : 180 ms typical, 240 ms maximum.
- Layer switch time : 75 ms.

The internal sorting algorithm takes the following rules into account in the following order.

- **Priority of the request** – High priority requests always take precedence, but too many of them can introduce too many extra seeks.
- **Time grouping** – Requests made within a certain time are grouped together to create a continuous reading operation on the disc for every time group. The default value is 2 seconds, but can be changed using the following cvar: sys_streaming_requests_grouping_time_period. Time grouping has a huge impact on the average completion time of the requests. It increases the time of a few otherwise quick reading requests, but drastically reduces the overall completion time because most of the streaming requests are coming from random places on the disc.
- **Actual offset on disc** – The actual disc offset is computed and used during the sorting. Files which have a higher offset get a higher priority, so it is important to organize the layout of the disc to reflect the desired streaming order.

For information regarding sorting, please refer to the source code in StreamAsyncFileRequest::ComputeSortKey(). The essential sorting code follows.

```cpp
void CAsyncIOFileRequest::ComputeSortKey(uint64 nCurrentKeyInProgress)
{
    // compute the disc offset (can be requested using CryPak)
    // group items by priority, then by snapped request time, then sort by disk offset
    m_nDiskOffset += m_nRequestedOffset;
}
```

1067
m_nTimeGroup = (uint64)(gEnv->pTimer->GetCurrTime() / max(1, g_cvars.sys_streaming_requests_grouping_time_period));
uint64 nPriority = m_ePriority;

int64 nDiskOffsetKB = m_nDiskOffset >> 10; // KB
m_nSortKey = (nDiskOffsetKB) | (((uint64)m_nTimeGroup) << 30) | (nPriority << 60);

Streaming Statistics

The streaming engine can be polled for streaming statistics using the GetStreamingStatistics() function.

Most of the statistics are divided into two groups, one collected during the last second, and another from the last reset (which usually happens during level loading). Statistics can also be forcibly reset during the game.

The SMediaTypeInfo struct gives information per IO input system (hard disk drive, optical, memory).

IStreamEngine.h

struct SMediaTypeInfo
{
    // stats collected during the last second
    float fActiveDuringLastSecond;
    float fAverageActiveTime;
    uint32 nBytesRead;
    uint32 nRequestCount;
    uint64 nSeekOffsetLastSecond;
    uint32 nCurrentReadBandwidth;
    uint32 nActualReadBandwidth; // only taking actual reading time into account

    // stats collected since last reset
    uint64 nTotalBytesRead;
    uint32 nTotalRequestCount;
    uint64 nAverageSeekOffset;
    uint32 nSessionReadBandwidth;
    uint32 nAverageActualReadBandwidth; // only taking actual read time into account
};

The SRequestTypeInfo struct gives information about each streaming request type, such as geometry, textures, and animations.

IStreamEngine.h

struct SRequestTypeInfo
{
    int nOpenRequestCount;
    int nPendingReadBytes;

    // stats collected during the last second
    uint32 nCurrentReadBandwidth;

    // stats collected since last reset
    uint32 nTotalStreamingRequestCount;
    uint64 nTotalReadBytes; // compressed data
    uint64 nTotalRequestDataSize; // uncompressed data
    uint32 nTotalRequestCount;
    uint32 nSessionReadBandwidth;

    float fAverageCompletionTime; // Average time it takes to fully complete a request
    float fAverageRequestCount; // Average amount of requests made per second
The following example shows global statistics that contain all the gathered data.

IStreamEngine.h

```c
struct SStatistics
{
    SMediaTypeInfo hddInfo;
    SMediaTypeInfo memoryInfo;
    SMediaTypeInfo opticalInfo;
    SRequestTypeInfo typeInfo[eStreamTaskTypeCount];

    uint32 nTotalSessionReadBandwidth; // Average read bandwidth in total from reset -
                                           // taking full time into account from reset
    uint32 nTotalCurrentReadBandwidth; // Total bytes/sec over all types and systems.

    int nPendingReadBytes;      // How many bytes still need to be read
    float fAverageCompletionTime; // Time in seconds on average takes to complete read
                                    // request.
    float fAverageRequestCount; // Average requests per second being done to streaming
                                 // engine
    int nOpenRequestCount;      // Amount of open requests

    uint64 nTotalBytesRead;     // Read bytes total from reset.
    uint32 nTotalRequestCount;  // Number of request from reset to the streaming engine.

    uint32 nDecompressBandwidth; // Bytes/second for last second

    int nMaxTempMemory;     // Maximum temporary memory used by the streaming system
};
```

Streaming Debug Information

Different types of debug information can be requested using the following CVar:

`sys_streaming_debug x`

Streaming and Levelcache Pak Files

As mentioned earlier, it is very important to minimize the seeks and seek distances when reading from an optical media drive. For this reason, the build system is designed to optimize the internal data layout for streaming.

The easiest and fastest approach is to not do any IO at all, but read the data from compressed data in memory. For this, small paks for startup and each level are created. These are loaded into memory during level loading. Some paks remain in memory until the end of the level. Others are only used to speed up the level loading. All small files and small read requests should ideally be diverted to these paks.

A special `RC_Job` build file is used to generate these paks: `Bin32/rc/RCJob_PerLevelCache.xml`. These paks are generated during a normal build pipeline. The internal management in the engine is done by the `CResourceManager` class, which uses the global `SystemEvents` to preload or unload the paks.

Currently, the following paks are loaded into memory during level loading (`sys_PakLoadCache`).

- `level.pak` – Contains all actual level data, and should not be touched after level loading anymore.
- `xml.pak`
- `dds0.pak` – Contains all lowest mips of all the textures in the level.
• **cfg.pak** and **cga.pak** – Only load when CGF streaming is enabled.

The following paks are cached into memory during the level load process (**sys_PakStreamCache**).

• **dds_cache.pak** - Contains all dds files smaller than 6 KB (except for dds.0 files).
• **cfg_cache.pak** - Contains all cfg files smaller than 32 KB (only when CGF streaming is enabled).

**Important**
Be sure that these paks are available. Without them, level loading can take up to a few minutes, and streaming performance is greatly reduced.

The information regarding all the resources of a level are stored in the **resourcelist.txt** and **auto_resourcelist.txt**. These files are generated by an automatic testing system which loads each level and executes a prerecorded playthrough on it. These resourcelist files are used during the build phase to generate the level paks.

All data not in these in memory paks is handled through IO on the optical drive or hard disk drive, and it is also best to reduce the amount of seeks here. This optimization phase is also performed during the build process using the resource compiler.

All the data which can be streamed is extracted from all the resource lists from all levels, and is removed from the default paks files (for example, **objects.pak**, **textures.pak**, **animations.pak**) and put into new optimized paks for streaming inside a streaming folder.

The creating of the streaming paks uses the following rules:

• **Split by extension**: Different extension files are put into different paks (for example, dds, caf, dba, cfg) so that files of the same type can be put close to each other. This enables them to be read in bursts. The paks are also used to increase the priority of certain file types during request sorting by using the disc offset.
• **Split by DDS type**: Different dds types are sorted differently to increase the priority of different types (for example, diffuse maps get higher priority than normal maps). The actual distance in the pak is used during the sorting of the request.
• **Split by DDS mip**: The highest mips are put into a separate pak file. They usually take more than 60% of the size of all the smaller mips and can then be streamed with a lower priority. This greatly reduces the average seek time required to read the smaller textures. The texture streaming system internally optimizes the reads to reflect these split texture data.
• **Sort alphabetically**: Default alphabetical sorting is required because some of the data (such as CGF’s during MP level loading), are loaded in alphabetical order. Changing this sort order can have a severe impact on the loading times.

The actual sorting code is hardcoded in the resource compiler, and can be found at: **Code\Tools\RC\ResourceCompiler\PakHelpers.cpp**.

**Important**
If you make changes to the sorting operator in the resource compiler, be sure to make the same changes to the texture streaming and streaming engine sorting operators.

**Single Thread IO Access and Invalid File Access**

It is very important that only a single thread access a particular IO device at one time. If multiple threads read from the same IO device concurrently, then the reading speed is more than halved, and it may take a number of seconds to read just a few kilobytes. This occurs because the IO reading head will partially read a few kilobytes for one thread, and then read another few kilobytes for another thread while always performing expensive seeks in between.
The solution is to exclusively read from StreamingIOThreads during gameplay. Lumberyard will by default show an **Invalid File Access** warning in the top left corner when reading data from the wrong thread, and will stall deliberately for three seconds to emulate the actual stall when reading from an optical drive.

## High Level Streaming Engine Usage

It is very easy to extend the current streaming functionality using the streaming engine. In this section, a small example class is presented that shows how to add a new streaming type.

First, create a class which derives from the IStreamCallback interface, which informs about streaming completion, and add some basic functionality to read a file. The file can either be read directly or use the streaming engine. When the data is read directly, it calls the `ProcessData` function to parse the loaded data. The function is also called from the async callback. Some processing can be performed here on the data if needed because it does not run on the main thread.

The default parameters are used when starting a reading request on the streaming engine. It is also possible to specify the final data storage to help reduce the number of dynamic allocations performed by the streaming engine.

The class also stores the read stream object in order to get information about the streaming request or to be able to cancel the request when the callback object is destroyed. The pointer is reset in the sync callback because after the call it will no longer be referenced by the streaming engine.

### CNewStreamingType

```cpp
#include
class CNewStreamingType : public IStreamCallback
{
public:
    CNewStreamingType() : m_pReadStream(0), m_bIsLoaded(false) {}
    ~CNewStreamingType()
    {
        if (m_pReadStream)
            m_pReadStream->Abort();
    }

    // Start reading some data
    bool ReadFile(const char* acFilename, bool bUseStreamingEngine)
    {
        if (bUseStreamingEngine)
        {
            StreamReadParams params;
            params.dwUserData = eLoadFullData;
            params.ePriority = estpNormal;
            params.nSize = 0; // read the full file
            params.pBuffer = NULL; // don't provide any buffer, but copy data when streaming is done

            m_pReadStream = g_pISystem->GetStreamEngine()->StartRead(eStreamTaskTypeNewType, acFilename, this, &params);
        }
        else
        {
            // old way of reading file in a sync way (blocking call!)
            const char* acData = 0;
            size_t stSize = 0;

            .. read file directly using CryPak or fopen/fread

            ProcessData(acData, stSize);
            m_bIsLoaded = true;
        }
    }
};
```
Text Localization and Unicode Support

Because games are typically localized to various languages, your game might have to use text data for many languages.

This document provides programming-related information regarding localization, including localization information specific to Lumberyard.

**Terminology**

The following table provides brief descriptions of some important terms related to localization and text processing.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>character</td>
<td>A unit of textual data. A character can be a glyph or formatting indicator. Note that a glyph does not necessarily form a single visible unit. For example, a diacritical mark [´] and the letter [a] are separate glyphs (and characters), but can be overlaid to form the character [â].</td>
</tr>
</tbody>
</table>
What encoding to use?

Since there are many methods of encoding text, the question that should be asked when dealing with even the smallest amount of text is, “In what encoding is this stored?” This is an important question because decoding a sequence of code-units in the wrong way will lead to encoding errors, or even worse, to valid decoding that yields the wrong content.

The following table describes some common encodings.

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Code-unit size</th>
<th>Code-point size</th>
<th>Maps the entire UCS space</th>
<th>Trivial to encode/decode</th>
<th>Immune to byte-order differences</th>
<th>Major users</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>7 bits</td>
<td>1 byte</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>Many English-only apps</td>
</tr>
<tr>
<td>UTF-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UTF-16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UTF-32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Because there is no single "best" encoding, you should always consider the scenario in which it will be used when choosing one.

Historically, different operating systems and software packages have chosen different sets of supported encodings. Even C++ follows different conventions on different operating systems. For example, the "wide character" wchar_t is 16-bits on Windows, but 32-bits on Linux.

Because Lumberyard products can be used on many operating systems and in many languages, full UCS coverage is desirable. The follow table presents some conventions used in Lumberyard:

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Code-unit size</th>
<th>Code-point size</th>
<th>Maps the entire UCS space</th>
<th>Trivial to encode/decode</th>
<th>Immune to byte-order differences</th>
<th>Major users</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ANSI) code-page</td>
<td>8 bits</td>
<td>varies, usually 1 byte</td>
<td>no</td>
<td>varies, usually yes</td>
<td>yes</td>
<td>Older OS functions</td>
</tr>
<tr>
<td>UTF-8</td>
<td>8 bits</td>
<td>1 to 4 bytes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>Most text on the internet, XML</td>
</tr>
<tr>
<td>UTF-16</td>
<td>16 bits</td>
<td>2 to 4 bytes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>Windows &quot;wide&quot; API, Qt</td>
</tr>
<tr>
<td>UCS-2</td>
<td>16 bits</td>
<td>2 bytes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>None (replaced with UTF-16)</td>
</tr>
<tr>
<td>UTF-32/UCS-4</td>
<td>32 bits</td>
<td>4 bytes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>Linux &quot;wide&quot; API</td>
</tr>
</tbody>
</table>

General principles
• Avoid using non-ASCII characters in source code. Consider using escape sequences if a non-ASCII literal is required.

• Avoid using absolute paths. Only paths that are under developer control should be entered. If possible, use relative ASCII paths for the game folder, root folder, and user folder. When this is not possible, carefully consider non-ASCII contents that may be under a user's control, such as those in the installation folder.

How does this affect me when writing code?

Since single-byte code-units are common (even in languages that also use double-byte code-units), single-byte string types can be used almost universally. In addition, since Lumberyard does not use ANSI code-pages, all text must be either ASCII or UTF-8.

The following properties hold for both ASCII and UTF-8.

• The NULL-byte (integral value 0) only occurs when a NULL-byte is intended (UTF-8 never generates a NULL-byte as part of multi-byte sequences). This means that C-style null-terminated strings act the same, and CRT functions like `strlen` will work as expected, except that it counts code-units, not characters.

• Code-points in the ASCII range have the same encoded value in UTF-8. This means that you can type English string literals in code and treat them as UTF-8 without conversion. Also, you can compare characters in the ASCII range directly against UTF-8 content (that is, when looking for an English or ASCII symbol sub-string).

• UTF-8 sequences (containing zero or more entire code-points) do not carry context. This means they are safe to append to each other without changing the contents of the text.

The difference between position and length in code-units (as reported through `string::length()`, `strlen()`, and similar functions) and their matching position and length in code-points is largely irrelevant. This is because the meaning of the sequence is typically abstract, and the meaning of the bytes matters only when the text is interpreted or displayed. However, keep in mind the following caveats.

• **Splitting strings** – When splitting a string, it's important to do one of the following.
  1. Recombine the parts in the same order after splitting, without interpreting the splitted parts as text (that is, without chunking for transmission).
  2. Perform the split at a boundary between code-points. The positions just before and just after any ASCII character are always safe.

• **API boundaries** – When an API accepts or returns strings, it's important to know what encoding the API uses. If the API doesn't treat strings as opaque (that is, interprets the text), passing UTF-8 may be problematic for APIs that accept byte-strings and interpret them as ASCII or ANSI. If no UTF-8 API is available, prefer any other Unicode API instead (UTF-16 or UTF-32). As a last resort, convert to ASCII, but understand that the conversion is lossy and cannot be recovered from the converted string. Always read the documentation of the API to see what text encoding it expects and perform any required conversion. All UTF encodings can be losslessly converted in both directions, so finding any API that accepts a UTF format gives you a way to use UTF encoding.

• **Identifiers** – When using strings as a "key" in a collection or for comparison, avoid using non-ASCII sequences as keys, as the concept of "equality" of UTF is complex due to normalization forms and locale-dependent rules. However, comparing UTF-8 strings byte-by-byte is safe if you only care about equality in terms of code-points (since code-point to code-unit mapping is 1:1).

• **Sorting** – When using strings for sorting, keep in mind that locale-specific rules for the order of text are complex. It's fine to let the UI deal with this in many cases. In general, make no assumptions of how a set of strings will be sorted. However, sorting UTF-8 strings as if they were ASCII will actually sort them by code-point. This is fine if you only require an arbitrary fixed order for `std::map` look-
up, but displaying contents in the UI in this order may be confusing for end-users that expect another ordering.

In general, avoid interpreting text if at all possible. Otherwise, try to operate on the ASCII subset and treat all other text parts as opaque indivisible sequences. When dealing with the concept of "length" or "size", try to consider using in code-units instead of code-points, since those operations are computationally cheaper. In fact, the concept of the "length" of Unicode sequences is complex, and there is a many-to-many mapping between code-points and what is actually displayed.

How does this affect me when dealing with text assets?

In general, always:

- Store text assets with UTF-8 encoding.
- Store with Unicode NFC (Normalization Form C). This is the most common form of storage in text editing tools, so it's best to use this form unless you have a good reason to do otherwise.
- Store text in the correct case (that is, the one that will be displayed). Case-conversion is a complex topic in many languages and is best avoided.

Utilities provided in CryCommon

Lumberyard provides some utilities to make it easy to losslessly and safely convert text between Unicode encodings. In-depth technical details are provided in the header files that expose the UnicodeFunctions.h and UnicodeIterator.h utilities.

The most common use cases are as follows.

```c++
string utf8;
wstring wide;
Unicode::Convert(utf8, wide); // Convert contents of wide string and store into UTF-8
string
Unicode::Convert(wide, utf8); // Convert contents of UTF-8 string to wide string

string ascii;
Unicode::Convert<Unicode::eEncoding_ASCII, Unicode::eEncoding_UTF8>(ascii, utf8); // Convert UTF-8 to ASCII (lossy!)
```

**Important**
The above functions assume that the input text is already validly encoded. To guard against malformed user input or potentially broken input, consider using the `Unicode::ConvertSafe` function.

Further reading

For an introduction to Unicode, see The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!).

For official information about Unicode, see The Unicode Consortium.
CryLog

CryLog Logging Functionality

You can log in Lumberyard by using the following global functions.

- CryLog (eMessage)
- CryLogAlways (eAlways)
- CryError (eError)
- CryWarning (eWarning)
- CryComment (eComment)

If more control is required, the ILog interface can be used directly by using the following syntax.

```cpp
gEnv->pLog->LogToFile("value %d",iVal);
```

Verbosity Level and Coloring

You can control the verbosity of logging with the console variables log_Verbosity and log_FileVerbosity.

The following table shows the levels of verbosity and color convention. In the console, warnings appear in yellow, and errors appear in red.

<table>
<thead>
<tr>
<th>Message</th>
<th>verbosity 0</th>
<th>verbosity 1</th>
<th>verbosity 2</th>
<th>verbosity 3</th>
<th>verbosity 4</th>
<th>Color in console</th>
</tr>
</thead>
<tbody>
<tr>
<td>eAlways</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>eErrorAlways</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>red</td>
</tr>
<tr>
<td>eWarningAlways</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>yellow</td>
</tr>
<tr>
<td>eInput</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>eError</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>red</td>
</tr>
<tr>
<td>eWarning</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>yellow</td>
</tr>
<tr>
<td>eMessage</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>eComment</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Key

- X – the message type is logged to the console or file
- ? – some special logic is involved
Tip
Full logging (to console and file) can be enabled by using log_Verbosity 4.

Log Files

The following log file sources write to the log files indicated.

<table>
<thead>
<tr>
<th>Source</th>
<th>Log file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lumberyard Editor</td>
<td>Editor.log</td>
</tr>
<tr>
<td>Game</td>
<td>game.log (default)</td>
</tr>
<tr>
<td>Error messages</td>
<td>Error.log</td>
</tr>
</tbody>
</table>

Console Variables

The following console variables relate to logging.

log_IncludeTime

Toggles time stamping of log entries.
Usage: log_IncludeTime [0/1/2/3/4/5]
0=off (default)
1=current time
2=relative time
3=current+relative time
4=absolute time in seconds since this mode was started
5=current time+server time

ai_LogFileVerbosity

None = 0, progress/errors/warnings = 1, event = 2, comment = 3

log_Verbosity DUMPTODISK

defines the verbosity level for log messages written to console
-1=suppress all logs (including eAlways)
0=suppress all logs(except eAlways)
1=additional errors
2=additional warnings
3=additional messages
4=additional comments