AWS Mobile SDK iOS Developer Guide

Release 0.0.3

Amazon Web Services

Apr 20, 2017

Contents

Welcome to the *AWS Mobile SDK for iOS Developer Guide*. This guide will help you start developing iOS applications using Amazon Web Services.

If you're new to the AWS Mobile SDK, you'll probably want to look first at *What Is the AWS SDK for iOS?* and *Getting Started with the AWS Mobile SDK for iOS*. These topics explain what the AWS Mobile SDK includes, how to set up the SDK, and how to get started using AWS services from an iOS application.

After you've set up the SDK, you can start working with the mobile clients for Amazon Web Services. The mobile clients are described in the service-specific topics. No matter which services you're using, you should provide AWS credentials to your app via Amazon Cognito. The Amazon Cognito credential provider is discussed briefly in the various mobile service topics, and at length in *Amazon Cognito for iOS*.

CHAPTER 1

What Is the AWS SDK for iOS?

The AWS SDK for iOS is an open-source software development kit, distributed under an Apache Open Source license. The SDK for iOS provides a library, code samples, and documentation to help developers build connected mobile applications using AWS. This guide explains how to get started with the SDK for iOS, how to work with related mobile services such as Amazon Cognito and Amazon Mobile Analytics, and how to use Amazon Web Services effectively in a mobile app.

1.1 About the AWS Mobile Services

The AWS Mobile SDKs include client-side libraries for working with Amazon Web Services. These client libraries provide high-level, mobile-optimized interfaces to services such as Amazon DynamoDB, Amazon Simple Storage Service, and Amazon Kinesis.

The Mobile SDKs also include clients for Amazon Cognito; and Amazon Mobile Analytics—web services designed specifically for use by mobile developers.

1.1.1 Amazon Cognito

Amazon Cognito facilitates the delivery of scoped, temporary credentials to mobile devices or other untrusted environments, and it uniquely identifies a device or user and supplies the user with a consistent identity throughout the lifetime of an application. The Amazon Cognito Sync service enables cross-device syncing of application-related user data. Cognito also persists data locally, so that it's available even if the device is offline.

After you've set up the SDK, you can start using Amazon Cognito by following the instructions at *Amazon Cognito for iOS* and *Amazon Cognito Sync: Sync User Data*.

1.1.2 Amazon Simple Storage Service (S3)

Amazon Simple Storage Service (S3) provides secure, durable, highly-scalable object storage in the cloud. Using the AWS Mobile SDK, you can directly access Amazon S3 from your mobile app.

After you've set up the SDK, you can start using Amazon S3 by following the instructions at *Amazon S3: Store and Retrieve Files and Data*.

1.1.3 Amazon DynamoDB

Amazon DynamoDB is a fast, highly scalable, highly available, cost-effective, nonrelational database service. DynamoDB removes traditional scalability limitations on data storage while maintaining low latency and predictable performance.

After you've set up the SDK, you can start using Amazon DynamoDB by following the instructions at *Amazon DynamoDB: Store and Retrieve Data*.

1.1.4 Amazon Kinesis

Amazon Kinesis is a fully managed service for real-time processing of streaming data at massive scale.

After you've set up the SDK, you can start using Amazon Kinesis by following the instructions at *Amazon Kinesis and Amazon Kinesis Firehose: Process Streaming Data*.

1.1.5 AWS Lambda

AWS Lambda is a compute service that runs your code in response to requests or events and automatically manages the compute resources for you, making it easy to build applications that respond quickly to new information.

After you've set up the SDK, you can start using AWS Lambda by following the instructions at *AWS Lambda: Execute Code On Demand*.

1.1.6 Amazon Lex

Amazon Lex is an AWS service for building voice and text conversational interfaces into applications. With Amazon Lex, the same deep learning engine that powers Amazon Alexa is now available to any developer, enabling you to build sophisticated, natural language chatbots into your new and existing applications.

After you've set up the SDK, you can start using Amazon Lex by following the instructions at *Amazon Lex: Use Natural Language to Trigger Business Workflows*.

1.1.7 Amazon Polly

Amazon Polly is a service that turns text into lifelike speech, making it easy to develop mobile applications that use high-quality speech to increase engagement and accessibility. With Amazon Polly you can quickly build speech-enabled apps that work in multiple geographies.

After you've set up the SDK, you can start using Amazon Polly by following the instructions at *Amazon Polly: Text to Speech Conversion*.

1.1.8 Amazon Pinpoint

Using Amazon Pinpoint, you can create push notification campaigns that provide your users with timely, relevant, personalized information to encourage them to keep using your mobile app. You can use push

notification campaigns to increase app awareness, downloads, and launches; build customer loyalty; and ultimately boost your mobile revenues.

After you've set up the SDK, you can start using Amazon Pinpoint by following the instructions at *Amazon Pinpoint: Create Push Notification Campaigns*.

1.1.9 Amazon Mobile Analytics

Amazon Mobile Analytics, which is now being rolled into Amazon Pinpoint, lets you collect, visualize, and understand app usage for your mobile apps. Reports are available for metrics on active users, sessions, retention, in-app revenue, and custom events, and can be filtered by platform and date range.

Amazon Pinpoint service offers analytics plus allows you to act programmatically on the data you gather through customizable push user engagement. Existing Amazon Mobile Analytics objects/configurations are automatically ported to Amazon Pinpoint. For the time being you can still use Amazon Mobile Analytics by following the instructions at *Amazon Mobile Analytics: Try Amazon Pinpoint*.

1.2 About the SDK for iOS

The AWS SDK for iOS includes:

- *AWS iOS libraries* – APIs that hide much of the lower-level plumbing of the web service interface, including authentication, request retries, and error handling. Each service has its own library, so you can include libraries for only the services you need.
- *Code samples* – Practical examples of using the libraries to build applications.
- *Documentation* – Reference documentation for the AWS SDK for iOS.

The SDK is distributed as a `.zip` file containing the following:

- `LICENSE`
- `NOTICE`
- `README.md`
- **frameworks/**
  - `AWSCore.framework`
  - `AWSAutoScaling.framework`
  - `AWSCloudWatch.framework`
  - `AWSDynamoDB.framework`
  - `AWSEC2.framework`
  - `AWSElasticLoadBalancing.framework`
  - `AWSKinesis.framework`
  - `AWSLambda.framework`

  - `AWSLex.framework`
  - `AWSMachineLearning.framework`
  - `AWSMobileAnalytics.framework`
  - `AWSPinpoint.framework`
  - `AWSPolly.framework`
  - `AWSS3.framework`
  - `AWSSES.framework`
  - `AWSSimpleDB.framework`
  - `AWSSNS.framework`
  - `AWSSQS.framework`
- **extras/**
  - `AWSCognito.framework` - The framework for Amazon Cognito sync.
  - `Amazon Software License.txt`
  - `NOTICE`
- **documentation/** – Contains documentation, including a docset, for the AWS SDK for iOS.
- **samples/** – Contains an HTML document that links to samples, which are named based on the services that they demonstrate.
- **src/** – Contains an HTML document that links to source, which contains the implementation and header files for the AWS iOS libraries.

CHAPTER 2

Getting Started with the AWS Mobile SDK for iOS

The AWS SDK for iOS provides the libraries, samples and, documentation needed to call AWS services from iOS apps. This Getting Started section covers the topics that are common to iOS apps that utilize AWS services, including:

2.1 Set Up the SDK for iOS

To get started with the AWS Mobile SDK for iOS you can set up the SDK and build a new project, or you can integrate the SDK in an existing project. You can also run the samples to get a sense of how the SDK works.

To use the SDK, install the following on your development machine:

- Xcode 7 or later
- iOS 8 or later

You can view the source code in the AWS Mobile SDK for iOS GitHub repository.

2.1.1 Include the AWS Mobile SDK for iOS in an Existing Application

The samples included with the SDK are standalone projects that are already set up. You can also integrate the SDK into your own existing project. Choose one of the following three ways to import the SDK into your project:

- CocoaPods
- Carthage
- Dynamic Frameworks

Note: Importing the SDK in multiple ways loads duplicate copies of the SDK into the project and causes compiler errors.

CocoaPods

1. The AWS Mobile SDK for iOS is available through CocoaPods. Install CocoaPods by running the following commands from the folder containing your projects `*.xcodeproj` file.

   $ **gem install cocoapods**

   ..note:: Depending on your system settings, you may need to run the command as administrator using *sudo*, as follows:

   $ **sudo gem install cocoapods**

   $ **pod setup**

   $ **pod init**

2. In your project directory (the directory where your `*.xcodeproj` file is), open the empty text file named `Podfile` (without a file extension) and add the following lines to the file. Replace `myAppName` with your app name. You can also remove pods for services that you don't use. For example, if you don't use *AWSAutoScaling*, remove or do not include the `AWSAutoScaling` pod.

```
source 'https://github.com/CocoaPods/Specs.git'

platform :ios, '8.0'
use_frameworks!

target :'myAppName' do
    pod 'AWSAutoScaling'
    pod 'AWSCloudWatch'
    pod 'AWSCognito'
    pod 'AWSCognitoIdentityProvider'
    pod 'AWSDynamoDB'
    pod 'AWSEC2'
    pod 'AWSElasticLoadBalancing'
    pod 'AWSIoT'
    pod 'AWSKinesis'
    pod 'AWSLambda'
    pod 'AWSLex'
    pod 'AWSMachineLearning'
    pod 'AWSMobileAnalytics'
    pod 'AWSPinpoint'
    pod 'AWSPolly'
    pod 'AWSRekognition'
    pod 'AWSS3'
    pod 'AWSSES'
    pod 'AWSSimpleDB'
    pod 'AWSSNS'
    pod 'AWSSQS'
end
```

3. Run the following command:

   $ **pod install**

4. Open `*.xcworkspace` with Xcode and start using the SDK.

> **Note:** Do not open `*.xcodeproj`. Opening this project file instead of a workspace results in an error.

Carthage

1. Install the latest version of Carthage.
2. Add the following to your *Cartfile*:

```
github "aws/aws-sdk-ios"
```

3. Run the following command:

   $ **carthage update**

4. With your project open in Xcode, choose your **Target**. In the **General** tab, find **Embedded Binaries**, then choose the **+** button.
5. Choose the **Add Other** button, navigate to the `AWS<#ServiceName#>.framework` files under **Carthage > Build > iOS** and select `AWSCore.framework` and the other service frameworks you require. Do not select the **Destination: Copy items if needed** checkbox when prompted.
   - `AWSCore.framework`
   - `AWSAutoScaling.framework`
   - `AWSCloudWatch.framework`
   - `AWSCognito.framework`
   - `AWSCognitoIdentityProvider.framework`
   - `AWSDynamoDB.framework`
   - `AWSEC2.framework`
   - `AWSElasticLoadBalancing.framework`
   - `AWSIoT.framework`
   - `AWSKinesis.framework`
   - `AWSLambda.framework`
   - `AWSLex.framework`
   - `AWSMachineLearning.framework`
   - `AWSMobileAnalytics.framework`
   - `AWSPinpoint.framework`
   - `AWSPolly.framework`
   - `AWSRekognition.framework`
   - `AWSS3.framework`

- AWSSES.framework
- AWSSimpleDB.framework
- AWSSNS.framework
- AWSSQS.framework

6. Under the **Build Phases** tab in your **Target**, choose the **+** button on the top left and then select **New Run Script Phase**.

Setup the build phase as follows. Make sure this phase is below the **Embed Frameworks** phase.

```
Shell /bin/sh

bash "${BUILT_PRODUCTS_DIR}/${FRAMEWORKS_FOLDER_PATH}/AWSCore.
↪framework/strip-frameworks.sh"

Show environment variables in build log: Checked
Run script only when installing: Not checked

Input Files: Empty
Output Files: Empty
```

Frameworks

1. Download the SDK from http://aws.amazon.com/mobile/sdk. The SDK is stored in a compressed file archive named aws-ios-sdk-#.#.#, where '#.#.#' represents the version number. For version 2.5.0, the filename is aws-ios-sdk-2.5.0.
2. With your project open in Xcode, choose your **Target**. Under the **General** tab, find **Embedded Binaries** and then choose the **+** button.
3. Choose **Add Other**. Navigate to the AWS<#ServiceName#>.framework files and select AWSCore.framework and the other service frameworks you require. Select the **Destination: Copy items if needed** checkbox when prompted.

   - AWSCore.framework
   - AWSAutoScaling.framework
   - AWSCloudWatch.framework
   - AWSCognito.framework
   - AWSCognitoIdentityProvider.framework
   - AWSDynamoDB.framework
   - AWSEC2.framework
   - AWSElasticLoadBalancing.framework
   - AWSIoT.framework
   - AWSKinesis.framework
   - AWSLambda.framework

- AWSLex.framework
- AWSMachineLearning.framework
- AWSMobileAnalytics.framework
- AWSPinpoint.framework
- AWSPolly.framework
- AWSRekognition.framework
- AWSS3.framework
- AWSSES.framework
- AWSSimpleDB.framework
- AWSSNS.framework
- AWSSQS.framework

4. Under the **Build Phases** tab in your **Target**, click the **+** button on the top left and then select **New Run Script Phase**.
5. Setup the build phase as follows. Make sure this phase is below the *Embed Frameworks* phase.

```
Shell /bin/sh

bash "${BUILT_PRODUCTS_DIR}/${FRAMEWORKS_FOLDER_PATH}/AWSCore.
↪framework/strip-frameworks.sh"

Show environment variables in build log: Checked
Run script only when installing: Not checked

Input Files: Empty
Output Files: Empty
```

2.1.2 Update the SDK to a Newer Version

This section describes how to pick up changes when a new SDK is released.

CocoaPods Run the following command in your project directory. CocoaPods automatically picks up the changes.

```
$ pod update
```

Note: If your pod update command fails, delete `Podfile.lock` and `Pods/` and then run **`pod install`** to cleanly install the SDK.

Carthage Run the following command in your project directory. Carthage automatically updates your frameworks with the new changes.

```
$ carthage update
```

Frameworks

1. In Xcode select the following frameworks in **Project Navigator** and press the **delete** key. Then select **Move to Trash**:
   - `AWSCore.framework`
   - `AWSAutoScaling.framework`
   - `AWSCloudWatch.framework`
   - `AWSCognito.framework`
   - `AWSCognitoIdentityProvider.framework`
   - `AWSDynamoDB.framework`
   - `AWSEC2.framework`
   - `AWSElasticLoadBalancing.framework`
   - `AWSIoT.framework`
   - `AWSKinesis.framework`
   - `AWSLambda.framework`
   - `AWSLex.framework`
   - `AWSMachineLearning.framework`
   - `AWSMobileAnalytics.framework`
   - `AWSPinpoint.framework`
   - `AWSPolly.framework`
   - `AWSRekognition.framework`
   - `AWSS3.framework`
   - `AWSSES.framework`
   - `AWSSimpleDB.framework`
   - `AWSSNS.framework`
   - `AWSSQS.framework`
2. Follow the manual Frameworks installation process to include the new version of the SDK.

2.1.3 Preparing to Work with ATS

The App Transport Security (ATS) feature, in the iOS 9.0 SDK or later, might impact how your apps interact with some AWS services.

If you compile your apps with the iOS 9.0 SDK (or Xcode 7) or later, there are additional steps you must complete for your app to successfully connect with any AWS service your app calls. For more information, see Preparing Your App to Work with ATS.

2.1.4 AWS Credentials

We recommend using Amazon Cognito as your credential provider to access AWS services from your mobile app. Amazon Cognito provides a secure mechanism to access AWS services without having to embed credentials in your app. To learn more, see *Amazon Cognito for iOS*.

Alternatively, you can use AWS Identity and Access Management (IAM) in combination with the AWS Security Token Service AssumeRole API. If you choose IAM, ensure that your role's policy is minimally scoped so that it can only perform the required actions for the service being used.

2.1.5 Import and Call SDK APIs with AWS Credentials

This section is included to give an overview of how you can connect your app to AWS services. For details about calling a specific service, see the left hand menu.

1. Import the AWSCore header in the application delegate.

   Swift

   ```
   import AWSCore
   import AWSCognito
   ```

   Objective-C

   ```
   #import <AWSCore/AWSCore.h>
   #import <AWSCognito/AWSCognito.h>
   ```

   Amazon Cognito APIs provide AWS identity services, and are included because they are used in the implementation of most mobile app features through AWS.

2. Create a default service configuration and establish an AWS identity provider by adding the following code snippet in the `application:didFinishLaunchingWithOptions:` application delegate method.

   Swift

   ```
   let credentialProvider =␣
   ↪AWSCognitoCredentialsProvider(regionType: .USEast1,␣
   ↪identityPoolId: "YourIdentityPoolId")
   let configuration = AWSServiceConfiguration(region: .USEast1,␣
   ↪credentialsProvider: credentialProvider)
   AWSServiceManager.default().defaultServiceConfiguration =␣
   ↪configuration
   ```

   Objective-C

   ```
   AWSCognitoCredentialsProvider *credentialsProvider =␣
   ↪[[AWSCognitoCredentialsProvider alloc] initWithRegionType:
   ↪AWSRegionUSEast1
   identityPoolId:@"YourIdentityPoolId"];

   AWSServiceConfiguration *configuration =␣
   ↪[[AWSServiceConfiguration alloc] initWithRegion:
   ↪AWSRegionUSEast1 credentialsProvider:credentialsProvider];
   ```

```
AWSServiceManager.defaultServiceManager.
 ↪defaultServiceConfiguration = configuration;
```

The value for `YourIdentityPoolId` is specific to the Amazon Cognito identity pool you create. To learn more, see *Amazon Cognito for iOS*.

3. Include import statements for each AWS service your app will call.

Swift

```
import AWSS3
import AWSDynamoDB
import AWSSQS
import AWSSNS
...
```

Objective-C

```
#import <AWSCore/AWSCore.h>
#import <AWSS3/AWSS3.h>
#import <AWSDynamoDB/AWSDynamoDB.h>
#import <AWSSQS/AWSSQS.h>
#import <AWSSNS/AWSSNS.h>
...
```

4. Make a call to your AWS service. In the example below, the call is to Amazon S3 through the SDKs AWSS3TransferManger API.

Swift

```
let transferManager = AWSS3TransferManager.default()

let uploadRequest = AWSS3TransferManagerUploadRequest()
uploadRequest.bucket = "myBucket"
uploadRequest.key = "myTestFile.txt"
uploadRequest.body = uploadingFileURL
uploadRequest.contentLength = fileSize

transferManager.upload(uploadRequest).continueWith(executor:␣
 ↪AWSExecutor.mainThread(), block: { (task:AWSTask<AnyObject>
 ↪) -> Any? in
    // Do something with the response
})
```

Objective-C

```
AWSS3Transfermanager *transferManager = [AWSS3Transfermanager␣
 ↪defaultS3TransferManager];

AWSS3TransferManagerUploadRequest *uploadRequest =␣
 ↪[AWSS3TransferManagerUploadRequest new];
uploadRequest.bucket = yourBucket;
```

```
uploadRequest.key = yourKey;
uploadRequest.body = yourDataURL;
uploadRequest.contentLength = [NSNumber␣
↪numberWithUnsignedLongLong:fileSize];

[[transferManager upload:uploadRequest] continueWithBlock:^
↪id(AWSTask *task) {
    // Do something with the response
    return nil;
}];
```

Note: Most of the service client classes have a singleton method to get a default client, named with the convention of adding `default` to the framework name. `AWSS3TransferManager.default()` (Swift) or `defaultS3TransferManager` (Objective-C) are examples in the preceding code snippet.

This singleton method creates a service client with `defaultServiceConfiguration`, which you initialized in the application delegate during a preceding step in this section. The method maintains a strong reference to the client.

2.1.6 Logging

As of version 2.5.4 of this SDK, logging utilizes CocoaLumberjack, a flexible, fast, open source logging framework. It supports many capabilities including the ability to set logging level per output target, for instance, concise messages logged to the console and verbose messages to a log file.

CocoaLumberjack logging levels are additive such that when the level is set to verbose, all messages from the levels below verbose are logged. It is also possible to set custom logging to meet your needs. For more information, see CocoaLumberjack

Changing Logging Level

You can change the logging level to suit the phase of your development cycle by importing AWSCore and calling:

Swift `AWSDDLog.sharedInstance().logLevel = .verbose`

The following logging level options are available:

- `.off`
- `.error`
- `.warning`
- `.info`

- `.debug`
- `.verbose`

We recommend setting the log level to `.off` before publishing to the App Store.

Objective-C `[AWSDDLog sharedInstance].logLevel = AWSDDLogLevelVerbose;`

The following logging level options are available:

- `AWSDDLogLevelOff`
- `AWSDDLogLevelError`
- `AWSDDLogLevelWarning`
- `AWSDDLogLevelInfo`
- `AWSDDLogLevelDebug`
- `AWSDDLogLevelVerbose`

We recommend setting the log level to `AWSDDLogLevelOff` before publishing to the App Store.

Targeting Log Output

CocoaLumberjack can direct logs to file or used as a framework that integrates with the Xcode console.

To initialize logging to files, use the following code:

Swift

```
let fileLogger: AWSDDFileLogger = AWSDDFileLogger() // File␣
↪Logger
fileLogger.rollingFrequency = TimeInterval(60*60*24)  // 24 hours
fileLogger.logFileManager.maximumNumberOfLogFiles = 7
AWSDDLog.add(fileLogger)
```

Objective-C

```
AWSDDFileLogger *fileLogger = [[AWSDDFileLogger alloc] init]; //␣
↪File Logger
fileLogger.rollingFrequency = 60 * 60 * 24; // 24 hour rolling
fileLogger.logFileManager.maximumNumberOfLogFiles = 7;
[AWSDDLog addLogger:fileLogger];
```

To initialize logging to your Xcode console, use the following code:

Swift

```
AWSDDLog.add(AWSDDTTYLogger.sharedInstance()) // TTY = Xcode␣
↪console
```

Objective-C

```
[AWSDDLog addLogger:[AWSDDTTYLogger sharedInstance]]; // TTY =␣
↪Xcode console
```

To learn more, see CocoaLumberjack on GitHub.

2.1.7 Sample Apps

The AWS Mobile SDK for iOS includes sample apps that demonstrate common use cases.

Amazon Cognito Your User Pools Sample (Objective-C)

This sample demonstrates how sign up and sign in a user to display an authenticated portion of your app.

AWS services demonstrated:

- Amazon Cognito Pools
- Amazon Cognito Identity

Amazon Cognito Sync Sample (Swift, Objective-C)

This sample demonstrates how to securely manage and sync your mobile app data. It also demonstrates how to create unique identities using login providers including Facebook, Google, and Login with Amazon.

AWS services demonstrated:

- Amazon Cognito Sync
- Amazon Cognito Identity

Amazon DynamoDB Object Mapper Sample (Swift, Objective-C)

This sample demonstrates how to insert, update, delete, query items using DynamoDBObjectMapper.

AWS services demonstrated:

- Amazon DynamoDB
- Amazon Cognito Identity

Amazon S3 Transfer Utility Sample (Swift, Objective-C)

This sample demonstrates how to use the Amazon S3 TransferUtility to download / upload files.

AWS services demonstrated:

- Amazon S3
- Amazon Cognito Identity

Amazon SNS Mobile Push and Mobile Analytics Sample (Swift, Objective-C)

This sample demonstrates how to set up Amazon SNS mobile push notifications and to record events using Amazon Mobile Analytics.

AWS services demonstrated:

- Amazon SNS (mobile push notification)
- Amazon Mobile Analytics
- Amazon Cognito Identity

2.1.8 Install the Reference Documentation in Xcode

The AWS Mobile SDK for iOS includes documentation in the DocSets format that you can view within Xcode. The easiest way to install the documentation is to use the macOS terminal.

To install the DocSet for Xcode

Open the macOS terminal and go to the directory containing the expanded archive. For example:

```
$ cd ~/Downloads/aws-ios-sdk-2.5.0
```

Note: Replace `2.5.0` in the preceding example with the version number of the AWS Mobile SDK for iOS that you downloaded.

Create a directory called `~/Library/Developer/Shared/Documentation/DocSets`:

```
$ mkdir -p ~/Library/Developer/Shared/Documentation/DocSets
```

Copy (or move) `documentation/com.amazon.aws.ios.docset` from the SDK installation files to the directory you created in the previous step:

```
$ mv documentation/com.amazon.aws.ios.docset
~/Library/Developer/Shared/Documentation/DocSets/
```

If Xcode was running during this procedure, restart Xcode. To browse the documentation, go to **Help**, click **Documentation and API Reference**, and select **AWS Mobile SDK for iOS v2.0 Documentation** (where '2.0' is the appropriate version number).

2.2 Amazon Cognito for iOS

For your app to access AWS services and resources, it must facilitate getting an identity within AWS for each user. Use Amazon Cognito to create unique identities for your users. Amazon Cognito identities can be unauthenticated, or they can use a range of methods to sign in and become authenticated. For more information, see *Integrating Identity Providers*.

For information about Amazon Cognito Region availability, see AWS Service Region Availability.

2.2.1 Providing AWS Credentials

Most implementations of AWS services for mobile app features require identity management through Amazon Cognito. The following steps describe how to AWS credentials to your app users.

In this section:

1. Create an identity pool and roles

Take the following steps to create a new identity pool with *Auth* and *Unauth* roles.

1. Sign in to the Amazon Cognito console.
2. Choose *Manage Federated Identities*.
3. Choose *Create new identity pool*.
4. Type an *Identity pool name*.
5. Optional: Select *Enable access to unauthenticated identities*.
6. Choose *Create Pool*.
7. Choose *View Details* to review or edit the role names and default access policy JSON document for the identity pool you just created. Note the names of your *Auth* and *Unauth* roles. You will use them to enact access policy for the AWS resources you use.
8. Choose: *Allow*.
9. Choose the language of your app code in the *Platform* menu. Note the *identityPoolId* value in the sample code provided.

For more information, see *Identity Pools and IAM Roles*.

2. Add the AWS SDK for iOS to your project

Follow the steps in *Set Up the SDK for iOS*.

3. Import *AWScore* and Amazon Cognito APIs

Add the following imports to your project.

Swift

```
import AWSCore
import AWSCognito
```

Objective-C

```
#import <AWSCore/AWSCore.h>
#import <AWSCognito/AWSCognito.h>
```

4. Initialize the Amazon Cognito credentials provider

Use the following code, replacing the value of *YourIdentityPoolId* with the *identitPoolId* value you noted when you created your identity pool.

Swift

```
let credentialProvider =␣
↪AWSCognitoCredentialsProvider(regionType: .USEast1,␣
↪identityPoolId: "YourIdentityPoolId")
let configuration = AWSServiceConfiguration(region: .USEast1,␣
↪credentialsProvider: credentialProvider)
AWSServiceManager.default().defaultServiceConfiguration =␣
↪configuration
```

Objective-C

```
AWSCognitoCredentialsProvider *credentialsProvider =␣
↪[[AWSCognitoCredentialsProvider alloc] initWithRegionType:
↪AWSRegionUSEast1
identityPoolId:@"YourIdentityPoolId"];

AWSServiceConfiguration *configuration =␣
↪[[AWSServiceConfiguration alloc] initWithRegion:
↪AWSRegionUSEast1 credentialsProvider:credentialsProvider];

AWSServiceManager.defaultServiceManager.
↪defaultServiceConfiguration = configuration;
```

Note: If you created your identity pool before February 2015, you must reassociate your roles with your identity pool to use this constructor. To do so, open the Amazon Cognito console, select your identity pool, choose *Edit Identity Pool*, specify your authenticated and unauthenticated roles, and save the changes

---

5. Retrieve Amazon Cognito IDs and AWS Credentials

After the login tokens are set in the credentials provider, you can retrieve a unique Amazon Cognito identifier for your end user and temporary credentials that let the app access your

AWS resources.

Swift

```
let cognitoId = credentialsProvider.identityId
```

Objective-C

```
// Retrieve your Amazon Cognito ID.
NSString *cognitoId = credentialsProvider.identityId;
```

The unique identifier is available in the `identityId` property of the credentials provider object.

The *credentialsProvider* communicates with Amazon Cognito, retrieving a unique identifier for the user as well as temporary, limited privilege AWS credentials for the AWS Mobile SDK. The retrieved credentials are valid for one hour.

2.2.2 Identity Pools and IAM Roles

To use Amazon Cognito to incorporate sign-in through an external identity provider into your app, create an Amazon Cognito identity pool.

An identity in a pool gets access to the AWS resources used by your app by being assigned a role in AWS Identity and Access Management (IAM). The access level of an IAM role is defined by the policy that is attached to it. Typical roles for identity pools allow you to give different levels of access to authenticated (*Auth*)or signed in users, and unauthenticated (*Unauth*)users.

For more information on identity pools, see Amazon Cognito Identity: Using Federated Identities.

For more information on using IAM roles with Amazon Cognito, see IAM Roles in the *Amazon Cognito Developer Guide*.

2.2.3 Integrating Identity Providers

Amazon Cognito identities can be unauthenticated or use a range of methods to sign in and become authenticated, including:

- Federating with an external provider such as Google or Facebook
  - For external providers, a developer account and an application registered with the identity provider you want to use (Facebook, Google, or Amazon)
- Federating with a SAML Provider such as a Microsoft Active Directory instance
  - For SAML federation, the SAML federation metadata for the authenticating system
- Federating with your existing custom authentication provider using developer authenticated identities
- Creating your own AWS-managed identity provider using Amazon Cognito User Pool

Then, each time your mobile app interacts with Amazon Cognito, your user's identity is given a set of temporary credentials that give secure access to the AWS resources configured for your app.

For information see, **'External Identity Providers<http://docs.aws.amazon.com/cognito/devguide/identity/external-providers/>'_** in the *Amazon Cognito Developer Guide*.

Related Documentation

Amazon Cognito Sync: Sync User Data

Developer Authenticated Identities

2.3 Working with Asynchronous Tasks

To work with asynchronous operations without blocking the UI thread, the SDK provides two options:

- `completionHandler`, a streamlined class which offers a simple, common pattern for most API calls

and

- `AWSTask`, a class which is a renamed version of BFTask from the Bolts framework. AWSTasks gives advantages for more complex operations like chaining asynchronous requests.

  For complete documentation on Bolts, see the Bolts-ObjC repo.

2.3.1 Using completionHandler

Most simple asynchronous API method calls can use `completionHandler` to handle method callbacks. When an asynchronous method is complete, `completionHandler` returns two parts: a response object containing the method's return if the call was successful, or `nil` if failed; and an error object containing the `NSError` state when a call fails, or `nil` upon success.

Handling Asynchronous Method Returns with completionHandler

The following code shows typical usage of `completionHandler` using Amazon Kinesis Firehose as the example.

Swift

```
var firehose = AWSFirehose.default()
firehose.putRecord(AWSFirehosePutRecordInput(),␣
↪completionHandler: {(_ response: AWSFirehosePutRecordOutput?, _
↪ error: Error?) -> Void in
    if error != nil {
        //handle error
    }
    else {
        //handle response
    }
})
```

Objective-C

```
AWSFirehose *firehose = [AWSFirehose defaultFirehose];

[firehose putRecord:[AWSFirehosePutRecordInput new]␣
↪completionHandler:^(AWSFirehosePutRecordOutput* _Nullable␣
↪response, NSError * _Nullable error) {
       if(error){
        //handle error
       }else{
        //handle response
       }
    }];
```

2.3.2 Using AWSTask

An `AWSTask` object represents the result of an asynchronous method. Using `AWSTask`, you can wait for an asynchronous method to return a value, and then something with that value after it has returned. You can chain asynchronous requests instead of nesting them. This helps keep logic clean and code readable.

Handling Asynchronous Method Returns with AWSTask

The following code shows how to use `continueOnSuccessBlockWith:` and `continueWith:` to handle methods calls that return an `AWSTask` object.

Swift

```
let kinesisRecorder = AWSKinesisRecorder.default()

let testData = "test-data".data(using: .utf8)
kinesisRecorder?.saveRecord(testData, streamName: "test-stream-
↪name").continueOnSuccessWith(block: { (task:AWSTask<AnyObject>
↪) -> AWSTask<AnyObject>? in
    // Guaranteed to happen after saveRecord has executed and␣
↪completed successfully.
    return kinesisRecorder?.submitAllRecords()
}).continueWith(block: { (task:AWSTask<AnyObject>) -> Any? in
    if let error = task.error as? NSError {
        print("Error: \(error)")
        return nil
    }

    return nil
})
```

Objective-C

```
AWSKinesisRecorder *kinesisRecorder = [AWSKinesisRecorder␣
↪defaultKinesisRecorder];

NSData *testData = [@"test-data" dataUsingEncoding:
↪NSUTF8StringEncoding];
```

```
[[[kinesisRecorder saveRecord:testData
                  streamName:@"test-stream-name"]␣
↪continueWithSuccessBlock:^id(AWSTask *task) {
    return [kinesisRecorder submitAllRecords];
}] continueWithBlock:^id(AWSTask *task) {
    if (task.error) {
        NSLog(@"Error: %@", task.error);
    }
    return nil;
}];
```

The `submitAllRecords` call is made within the `continueOnSuccessWith` / `continueWithSuccessBlock:` because we want to run `submitAllRecords` after `saveRecord:streamName:` successfully finishes running. The `continueWith` and `continueOnSuccessWith` won't run until the previous asynchronous call finishes. In this example, `submitAllRecords` is guaranteed to see the result of `saveRecord:streamName:`.

Handling Errors with AWSTask

The `continueWith:` and `continueOnSuccessWith:` block calls work in similar ways. Both ensure that the previous asynchronous method finishes executing before the subsequent block runs. However, they have one important difference: `continueOnSuccessWith:` is skipped if an error occurred in the previous operation, but `continueWith:` is always executed.

For example, consider the following scenarios, which refer to the preceding code snippet above.

- `saveRecord:streamName:` succeeded and `submitAllRecords` succeeded.

  In this scenario, the program flow proceeds as follows:

  1. `saveRecord:streamName:` is successfully executed.
  2. `continueOnSuccessWith:` is executed.
  3. `submitAllRecords` is successfully executed.
  4. `continueWith:` is executed.
  5. Because `task.error` is nil, it doesn't log an error.
  6. Done.

- `saveRecord:streamName:` succeeded and `submitAllRecords` failed.

  In this scenario, the program flow proceeds as follows:

  1. `saveRecord:streamName:` is successfully executed.
  2. `continueOnSuccessWith` is executed.
  3. `submitAllRecords` is executed with an error.
  4. `continueWithBlock:` is executed.
  5. Because `task.error` is NOT nil, it logs an error from `submitAllRecords`.

6. Done.

- `saveRecord:streamName:` failed.

  In this scenario, the program flow proceeds as follows:

  1. `saveRecord:streamName:` is executed with an error.
  2. `continueOnSuccessWith:` is skipped and will NOT be executed.
  3. `continueWithBlock:` is executed.
  4. Because `task.error` is NOT nil, it logs an error from `saveRecord:streamName:`.
  5. Done.

Consolidated Error Logic with AWSTask

The preceding example consolidates error handling logic at the end of the execution chain for both methods called. It doesn't check for `task.error` in `continueOnSuccessBlockWith:`, but waits until the `continueWith:` block executes to do so. An error from either the `submitAllRecords` or the `saveRecord:streamName:` method will be printed.

Per Method Error Logic with AWSTask

The following code shows how to implement the same behavior, but makes error handling specific to each method. `submitAllRecords` is only called if `saveRecord:streamName` succeeds, however, in this case, the `saveRecord:streamName` call uses `continueWith:`, the block logic checks `task.error` and returns nil upon error. If that block succeeds then `submitAllRecords` is called using `continueWith:` in a block that also checks `task.error` for its own context.

Swift

```
let kinesisRecorder = AWSKinesisRecorder.default()

let testData = "test-data".data(using: .utf8)
kinesisRecorder?.saveRecord(testData, streamName: "test-stream-
↪name").continueWith(block: { (task:AWSTask<AnyObject>) ->␣
↪AWSTask<AnyObject>? in
    if let error = task.error as? NSError {
        print("Error from 'saveRecord:streamName:': \(error)")
        return nil
    }
    return kinesisRecorder?.submitAllRecords()
}).continueWith(block: { (task:AWSTask<AnyObject>) -> Any? in
    if let error = task.error as? NSError {
        print("Error from 'submitAllRecords': \(error)")
        return nil
    }

    return nil
})
```

Objective-C

```
AWSKinesisRecorder *kinesisRecorder = [AWSKinesisRecorder␣
↪defaultKinesisRecorder];

NSData *testData = [@"test-data" dataUsingEncoding:
↪NSUTF8StringEncoding];
[[[kinesisRecorder saveRecord:testData
    streamName:@"test-stream-name"] continueWithBlock:^
↪id(AWSTask *task) {
    if (task.error) {
        NSLog(@"Error from 'saveRecord:streamName:': %@", task.
↪error);
        return nil;
    }
    return [kinesisRecorder submitAllRecords];
}]continueWithBlock:^id(AWSTask *task) {
    if (task.error) {
        NSLog(@"Error from 'submitAllRecords': %@", task.
↪error);
    }
    return nil;
}];
```

Returning AWSTask or nil

Remember to return either an `AWSTask` object or `nil` in every usage of `continueWith:` and `continueOnSuccessWith:`. In most cases, Xcode provides a warning if there is no valid return present, but in some cases an undefined error can occur.

Executing Multiple Tasks with AWSTask

If you want to execute a large number of operations, you have two options: executing in sequence or executing in parallel.

In Sequence

You can submit 100 records to an Amazon Kinesis stream in sequence as follows:

Swift

```
var task = AWSTask<AnyObject>(result: nil)

for i in 0...100 {
    task = task.continueOnSuccessWith(block: { (task:AWSTask
↪<AnyObject>) -> AWSTask<AnyObject>? in
        return kinesisRecorder!.saveRecord(String(format:
↪"TestString-%02d", i).data(using: .utf8), streamName:
↪"YourStreamName")
```

```
    })
}

task.continueOnSuccessWith { (task:AWSTask<AnyObject>) -> AWSTask
↪<AnyObject>? in
    return kinesisRecorder?.submitAllRecords()
}
```

Objective-C

```
AWSKinesisRecorder *kinesisRecorder = [AWSKinesisRecorder␣
↪defaultKinesisRecorder];

AWSTask *task = [AWSTask taskWithResult:nil];
for (int32_t i = 0; i < 100; i++) {
    task = [task continueWithSuccessBlock:^id(AWSTask *task) {
        NSData *testData = [[NSString stringWithFormat:@
↪"TestString-%02d", i] dataUsingEncoding:NSUTF8StringEncoding];
        return [kinesisRecorder saveRecord:testData
                                streamName:@"test-stream-name"];
    }];
}

[task continueWithSuccessBlock:^id(AWSTask *task) {
    return [kinesisRecorder submitAllRecords];
}];
```

In this case, the key is to concatenate a series of tasks by reassigning `task`.

Swift

```
task.continueOnSuccessWith { (task:AWSTask<AnyObject>) -> AWSTask
↪<AnyObject>? in
```

Objective-C

```
task = [task continueWithSuccessBlock:^id(AWSTask *task) {
```

In Parallel

You can execute multiple methods in parallel by using `taskForCompletionOfAllTasks:` as follows.

Swift

```
var tasks = Array<AWSTask<AnyObject>>()
for i in 0...100 {
    tasks.append(kinesisRecorder!.saveRecord(String(format:
↪"TestString-%02d", i).data(using: .utf8), streamName:
↪"YourStreamName")!)
}
```

```
AWSTask(forCompletionOfAllTasks: tasks).
 ↪continueOnSuccessWith(block: { (task:AWSTask<AnyObject>) ->␣
 ↪AWSTask<AnyObject>? in
    return kinesisRecorder?.submitAllRecords()
}).continueWith(block: { (task:AWSTask<AnyObject>) -> Any? in
    if let error = task.error as? NSError {
        print("Error: \(error)")
        return nil
    }

    return nil
})
```

Objective-C

```
AWSKinesisRecorder *kinesisRecorder = [AWSKinesisRecorder␣
 ↪defaultKinesisRecorder];

NSMutableArray *tasks = [NSMutableArray new];
for (int32_t i = 0; i < 100; i++) {
    NSData *testData = [[NSString stringWithFormat:@"TestString-
 ↪%02d", i] dataUsingEncoding:NSUTF8StringEncoding];
    [tasks addObject:[kinesisRecorder saveRecord:testData
                                      streamName:@"test-stream-
 ↪name"]];
}

[[AWSTask taskForCompletionOfAllTasks:tasks]␣
 ↪continueWithSuccessBlock:^id(AWSTask *task) {
    return [kinesisRecorder submitAllRecords];
}];
```

In this example you create an instance of `NSMutableArray`, put all of our tasks in it, and then pass it to `taskForCompletionOfAllTasks:`, which is successful only when all of the tasks are successfully executed. This approach may be faster, but it may consume more system resources. Also, some AWS services, such as Amazon DynamoDB, throttle a large number of certain requests. Choose a sequential or parallel approach based on your use case.

Executing a Block on the Main Thread with AWSTask

By default, `continueWithBlock:` and `continueWithSuccessBlock:` are executed on a background thread. But in some cases (for example, updating a UI component based on the result of a service call), you need to execute an operation on the main thread. To execute an operation on the main thread, you can use Grand Central Dispatch or `AWSExecutor`.

The following example shows the use of `dispatch_async(dispatch_get_main_queue(),^{...});` to execute a block on the main thread. For error handling, it creates a `UIAlertView` on the main thread when record submission fails.

Swift

```
let kinesisRecorder = AWSKinesisRecorder.default()

let testData = "test-data".data(using: .utf8)
kinesisRecorder?.saveRecord(testData, streamName: "test-stream-
↪name").continueOnSuccessWith(block: { (task:AWSTask<AnyObject>
↪) -> AWSTask<AnyObject>? in
    return kinesisRecorder?.submitAllRecords()
}).continueWith(block: { (task:AWSTask<AnyObject>) -> Any? in
    if let error = task.error as? NSError {
        DispatchQueue.main.async(execute: {
            let alertController = UIAlertView(title: "Error!",␣
↪message: error.description, delegate: nil, cancelButtonTitle:
↪"OK")
            alertController.show()
        })
        return nil
    }

    return nil
})
```

Objective-C

```
AWSKinesisRecorder *kinesisRecorder = [AWSKinesisRecorder␣
↪defaultKinesisRecorder];

NSData *testData = [@"test-data" dataUsingEncoding:
↪NSUTF8StringEncoding];
[[[kinesisRecorder saveRecord:testData
                   streamName:@"test-stream-name"]␣
↪continueWithSuccessBlock:^id(AWSTask *task) {
    return [kinesisRecorder submitAllRecords];
}] continueWithBlock:^id(AWSTask *task) {
    if (task.error) {
        dispatch_async(dispatch_get_main_queue(), ^{
            UIAlertView *alertView =
                [[UIAlertView alloc] initWithTitle:@"Error!"
                                           message:[NSString␣
↪stringWithFormat:@"Error: %@", task.error]
                                          delegate:nil
                                 cancelButtonTitle:@"OK"
                                 otherButtonTitles:nil];
            [alertView show];
        });
    }
    return nil;
}];
```

Another option is to use `AWSExecutor` as follows.

Swift

```
let kinesisRecorder = AWSKinesisRecorder.default()

let testData = "test-data".data(using: .utf8)
kinesisRecorder?.saveRecord(testData, streamName: "test-stream-
↪name").continueOnSuccessWith(block: { (task:AWSTask<AnyObject>
↪) -> AWSTask<AnyObject>? in
    return kinesisRecorder?.submitAllRecords()
}).continueWith(executor: AWSExecutor.mainThread(), block: {␣
↪(task:AWSTask<AnyObject>) -> Any? in
    if let error = task.error as? NSError {
        let alertController = UIAlertView(title: "Error!",␣
↪message: error.description, delegate: nil, cancelButtonTitle:
↪"OK")
        alertController.show()
        return nil
    }

    return nil
})
```

Objective-C

```
AWSKinesisRecorder *kinesisRecorder = [AWSKinesisRecorder␣
↪defaultKinesisRecorder];

NSData *testData = [@"test-data" dataUsingEncoding:
↪NSUTF8StringEncoding];
[[[kinesisRecorder saveRecord:testData streamName:@"test-stream-
↪name"]
          continueWithSuccessBlock:^id(AWSTask *task) {
      return [kinesisRecorder submitAllRecords];
}] continueWithExecutor:[AWSExecutor mainThreadExecutor]␣
↪withBlock:^id(AWSTask *task) {
    if (task.error) {
        UIAlertView *alertView =
            [[UIAlertView alloc] initWithTitle:@"Error!"
                       message:[NSString stringWithFormat:@"Error:
↪%@", task.error]
                       delegate:nil
                       cancelButtonTitle:@"OK"
                       otherButtonTitles:nil];
        [alertView show];
    }
    return nil;
}];
```

In this case, `withBlock:` (Objective-C) or `block:` (Swift) is executed on the main thread.

2.4 Preparing Your App to Work with ATS

If you use the iOS 9 SDK (or Xcode 7) or later, the Apple App Transport Security (ATS) feature might impact how your apps interact with some AWS services.

If your app targeted for iOS 9+ attempts to connect to an AWS service endpoint that does not yet meet all the ATS requirements, the connection may fail. The following sections provide instructions to determine if your app is affected, and what steps to take to mitigate the impact of ATS on your app.

2.4.1 Diagnosing ATS Conflicts

If your app stops working after being upgraded to Xcode 7 or later and iOS 9 or later, follow these steps to determine if it affected by ATS.

1. Turn on verbose logging of the AWS Mobile SDK for iOS by calling the following line in the `-application:didFinishLaunchingWithOptions:` application delegate.

   Swift

   ```
   @UIApplicationMain
   class AppDelegate: UIResponder, UIApplicationDelegate {

       var window: UIWindow?
       func application(application: UIApplication,␣
    ↪didFinishLaunchingWithOptions
       launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
           ...
           AWSLogger.default().logLevel = .verbose
           ...

           return true
       }
   }
   ```

   Objective-C

   ```
   #import <AWSCore/AWSCore.h>

   @implementation AppDelegate

   - (BOOL)application:(UIApplication *)application
   didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
       ...
       [AWSLogger defaultLogger].logLevel = AWSLogLevelVerbose;
       ...

       return YES;
   }

   @end
   ```

2. Run your app and make a request to an AWS service.

3. Search your log output for "SSL". The message containing: "An SSL error has occurred and a secure connection to the server cannot be made" indicates that your app is affected by the ATS changes.

```
2015-10-06 11:39:13.402 DynamoDBSampleSwift[14467:303540]␣
↪CFNetwork SSLHandshake failed (-9824)
2015-10-06 11:39:13.403 DynamoDBSampleSwift[14467:303540]␣
↪NSURLSession/NSURLConnection HTTP load failed␣
↪(kCFStreamErrorDomainSSL, -9824)
2015-10-06 11:39:13.569 DynamoDBSampleSwift[14467:303540]␣
↪CFNetwork SSLHandshake failed (-9824)
2015-10-06 11:39:13.569 DynamoDBSampleSwift[14467:303540]␣
↪NSURLSession/NSURLConnection HTTP load failed␣
↪(kCFStreamErrorDomainSSL, -9824)
Error: Error Domain=NSURLErrorDomain Code=-1200 "An SSL error␣
↪has occurred and a secure connection to the server cannot be␣
↪made." UserInfo={_kCFStreamErrorCodeKey=-9824,␣
↪NSLocalizedRecoverySuggestion=Would you like to connect to the␣
↪server anyway?, NSUnderlyingError=0x7fca343012f0 {Error␣
↪Domain=kCFErrorDomainCFNetwork Code=-1200 "(null)" UserInfo={_
↪kCFStreamPropertySSLClientCertificateState=0, _
↪kCFNetworkCFStreamSSLErrorOriginalValue=-9824, _
↪kCFStreamErrorDomainKey=3, _kCFStreamErrorCodeKey=-9824}},␣
↪NSLocalizedDescription=An SSL error has occurred and a secure␣
↪connection to the server cannot be made.,␣
↪NSErrorFailingURLKey=https://dynamodb.us-east-1.amazonaws.com/
↪, NSErrorFailingURLStringKey=https://dynamodb.us-east-1.
↪amazonaws.com/, _kCFStreamErrorDomainKey=3}
```

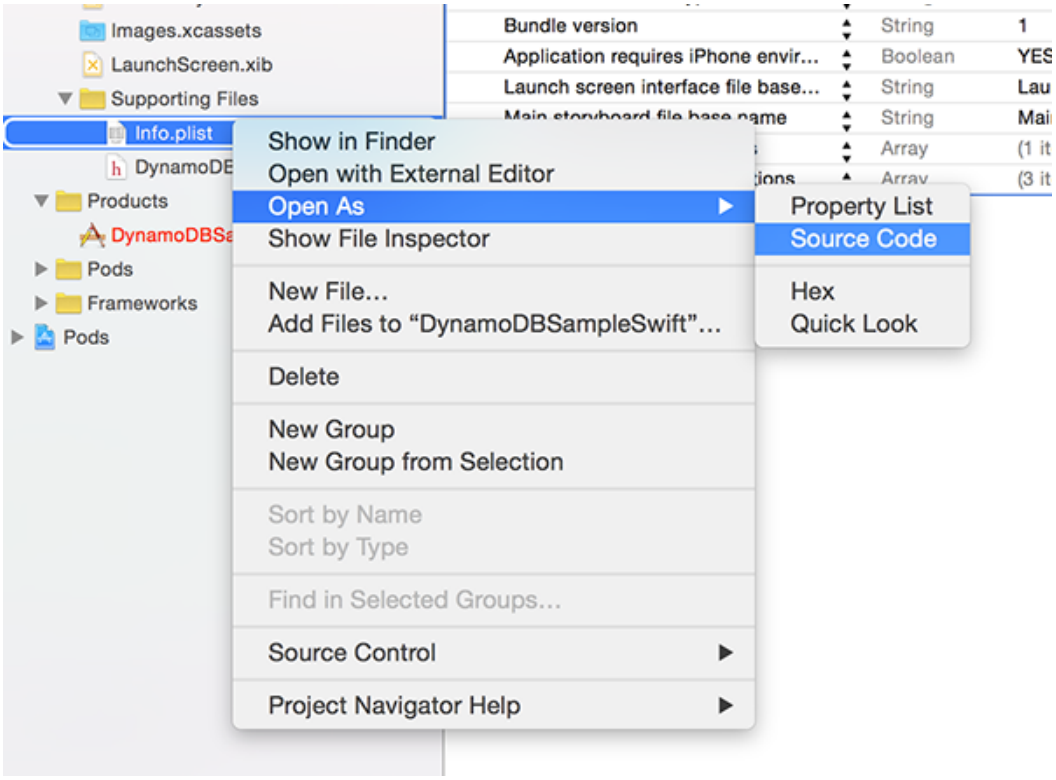
   If you cannot find the SSL handshake error message, it is possible that another problem caused your app to stop working. Some internal behaviors change with major operating system updates, and it is common for previously unseen issues to surface.

   If you are unable to resolve such issues, you can post code snippets, and steps to reproduce the issue on our forum or GitHub so that we can assist you in identifying the issue. Remember to include the versions of Xcode, iOS, and the AWS Mobile SDK.

2.4.2 Mitigating ATS Connection Issues

If you determine that your app is impacted by the diagnostic handshake error, you can configure the app to interact properly with the ATS feature by taking the following steps to add properties to your `Info.plist` file.

1. Locate your `Info.plist` and from the context menu select **Open As > Source Code**.



2. Copy and paste the following key as a direct child of the top level `<dict>` tag.

```
<plist version="1.0">
  <key>NSAppTransportSecurity</key>
  <dict>
      <key>NSExceptionDomains</key>
      <dict>
          <key>amazonaws.com</key>
          <dict>
                <key>NSThirdPartyExceptionMinimumTLSVersion</key>
                <string>TLSv1.0</string>
                <key>NSThirdPartyExceptionRequiresForwardSecrecy
↪</key>
                <false/>
                <key>NSIncludesSubdomains</key>
                <true/>
          </dict>
          <key>amazonaws.com.cn</key>
          <dict>
                <key>NSThirdPartyExceptionMinimumTLSVersion</key>
                <string>TLSv1.0</string>
                <key>NSThirdPartyExceptionRequiresForwardSecrecy
↪</key>
                <false/>
                <key>NSIncludesSubdomains</key>
                <true/>
          </dict>
      </dict>
  </dict>
```

```
  . . .
</plist>
```

After following these steps, your app should be able to access AWS endpoints while running on iOS 9 or later.

For more information about the AWS SDK for iOS see *What Is the AWS SDK for iOS?*.

For more information about other AWS SDKs see AWS Mobile SDK.

CHAPTER 3

Amazon S3: Store and Retrieve Files and Data

Amazon Simple Storage Service (S3) provides secure, durable, highly-scalable object storage in the cloud. Using the AWS Mobile SDK, you can directly access Amazon S3 from your mobile app.

In this section:

3.1 Amazon S3 Integration Setup

To integrate Amazon S3 features of the AWS SDK for iOS into an app, take the following steps.

Create and configure the following AWS services and policies.

1. Install the SDK

   Add the AWS SDK for iOS to your project and import the APIs you need, by following the steps in *Set Up the SDK for iOS*.

2. Configure credentials

   To use Amazon Cognito to create AWS identities and credentials that give your users access to your app's AWS resources, follow the steps in *Amazon Cognito for iOS*.

3. Create and configure an Amazon S3 bucket

   Amazon S3 stores your resources in buckets, which are AWS containers for objects. Buckets are created in specific regions. Each bucket must have a globally unique name.

   Create a bucket

   (a) Sign in to the Amazon S3 console.

   (b) Choose *Create Bucket*.

   (c) Type a bucket name, choose a region, and then choose *Create Bucket*.

   Grant Permissions

   Like most AWS service objects, Amazon S3 buckets have access policies attached to them that you can use to grant permissions for IAM entities, such as roles or individual identities. Take the following steps to grant the unauthenticated IAM role of your app's identity pool permissions to the bucket you created.

(a) Navigate to the Identity and Access Management console.

(b) Choose *Roles* in the left navigation pane.

(c) Type your identity pool name into the search box. Two roles are listed: one for unauthenticated users and one for authenticated users.

(d) Choose the role for unauthenticated users (it has *unauth* appended to your identity pool name).

(e) At the bottom of the *Permissions* tab, find the policy AWS attached when you created the role and choose *Create Role Policy*.

(f) Choose *Custom Policy*, and then choose *Select*.

(g) Enter a name in *Policy Name*, and then copy and paste the following policy statement into the *Policy Document* area.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["s3:*"],
            "Resource": ["arn:aws:s3:::examplebucket/"]
        }
    ]
}
```

(h) Choose *Apply Policy*.

This policy grants the user permissions for all actions in all objects in the specified bucket. For more information on granting access to Amazon S3, see Granting Access to an Amazon S3 Bucket.

Upload files from the console

The following steps describe how to manually upload the file used in this walk through to the bucket you have created.

(a) In the Amazon S3 console, navigate to your bucket.

(b) In the *Actions'drop down menu, choose :guilabel:'Upload*.

(c) Choose + *Add Files* and select a test file to upload. For this walk through, we'll assume you're uploading an image called `myImage.jpg`.

(d) With your test image selected, choose *Start Upload*.

3.1.1 Additional Resources

- Amazon Simple Storage Service Getting Started Guide
- Amazon Simple Storage Service API Reference
- Amazon Simple Storage Service Developer Guide

3.2 Amazon S3 TransferManager for iOS

On this page:

Amazon Simple Storage Service (S3)

Amazon Simple Storage Service (S3) provides secure, durable, highly-scalable object storage in the cloud. Using the AWS Mobile SDK for iOS, you can directly access Amazon S3 from your mobile app. For information about Amazon S3 regional availability, see AWS Service Region Availability.

TransferManager Features

Amazon S3 TransferManager class makes it easy to upload files to and download files from Amazon S3 while optimizing for performance and reliability. It hides the complexity of transferring files behind a simple API.

Whenever possible, uploads are broken into multiple pieces, so that several pieces are sent in parallel to provide better throughput. This approach enables more robust transfers, since an I/O error in an individual piece result in the SDK retransmitting only the faulty piece, not the entire transfer. `TransferManager` provides simple APIs to pause, resume, and cancel file transfers.

The following sections provide a step-by-step guide for getting started with Amazon S3 using the `TransferManager`.

You can also try out the Amazon S3 sample available in the AWSLabs GitHub repository.

Should I Use `TransferManager` or `TransferUtility`?

To choose which API best suits your needs, see *Should I Use TransferManager or TransferUtility?*.

3.2.1 Setup

To set your project up to use the `TransferManager` class, take the steps below.

1. Setup the SDK, Credentials and Services

Follow the steps in *Amazon S3 Integration Setup* install the AWS Mobile SDK for iOS and configure AWS credentials and permissions.

2. Import the SDK Amazon S3 APIs

Add the following import statements to your Xcode project.

Swift

```
import AWSS3
```

Objective-C

```
#import <AWSS3/AWSS3.h>
```

3. Create the `TransferManager` Client

Add the following code to create an *AWSS3TransferManager* client.

Swift

```
let transferManager = AWSS3TransferManager.default()
```

Objective-C

```
AWSS3TransferManager *transferManager =␣
↪[AWSS3TransferManager defaultS3TransferManager];
```

The *AWSS3TransferManager* class is an entry point to this SDK's high-level Amazon S3 APIs.

3.2.2 Transfer an Object

In this section:

Downloading a file from and uploading a file to a bucket, use the same coding pattern. An important difference is that *download:* does not succeed until the download is complete, blocking any flow that depends on that success. Upload returns immediately and can therefore be safely called on the main thread.

The steps to call `TransferManager` for a transfer are as follows.

1. Create an `AWSS3TransferManagerDownloadRequest`

The following code illustrates the three actions needed to create a download request:

- Create a destination/source location for the file. In this example, this is called `downloadingFileURL` / `uploadingFileURL`.
- Construct a request object using `AWSS3TransferManagerDownloadRequest`.
- Set three properties of the request object: the bucket name; the key (the name of the object in the bucket); and the download destination / upload source `downloadingFileURL` / `uploadingFileURL`.

Download

Swift

```
let downloadingFileURL = URL(fileURLWithPath:␣
↪NSTemporaryDirectory()).appendingPathComponent(
↪"myImage.jpg")

let downloadRequest =␣
↪AWSS3TransferManagerDownloadRequest()

downloadRequest.bucket = "myBucket"
downloadRequest.key = "myImage.jpg"
downloadRequest.downloadingFileURL = downloadingFileURL
```

Objective-C

```
NSString *downloadingFilePath = [NSTemporaryDirectory()␣
↪stringByAppendingPathComponent:@"myImage.jpg"];
NSURL *downloadingFileURL = [NSURL fileURLWithPath:
↪downloadingFilePath];

AWSS3TransferManagerDownloadRequest *downloadRequest =␣
↪[AWSS3TransferManagerDownloadRequest new];

downloadRequest.bucket = @"myBucket";
downloadRequest.key = @"myImage.jpg";
downloadRequest.downloadingFileURL = downloadingFileURL;
```

Upload

Swift

```
let uploadingFileURL = URL(fileURLWithPath: "your/file/
↪path/myTestFile.txt")

let uploadRequest = AWSS3TransferManagerUploadRequest()

uploadRequest.bucket = "myBucket"
uploadRequest.key = "myTestFile.txt"
uploadRequest.body = uploadingFileURL
```

Objective-C

```
NSURL *uploadingFileURL = [NSURL fileURLWithPath: "your/
↪file/path/myTestFile.txt"];

AWSS3TransferManagerUploadRequest *uploadRequest =␣
↪[AWSS3TransferManagerUploadRequest new];

uploadRequest.bucket = @"myBucket";
uploadRequest.key = @"myTestFile.txt";
uploadRequest.body = uploadingFileURL;
```

2. Pass the Request to the *download:* Method

Use the following code to pass the request to the *download: / upload:" method of the ‘‘TransferManager‘* client. The methods are asynchronous and returns an *AWSTask* object. Use a *continueWith* block to handle the method result. For more information about *AWSTask*, see awstask.

Download

Swift

```
transferManager.download(downloadRequest).
↪continueWith(executor: AWSExecutor.mainThread(), block:
↪ { (task:AWSTask<AnyObject>) -> Any? in

    if let error = task.error as? NSError {
        if error.domain ==␣
↪AWSS3TransferManagerErrorDomain, let code =␣
↪AWSS3TransferManagerErrorType(rawValue: error.code) {
            switch code {
            case .cancelled, .paused:
                break
            default:
                print("Error downloading:␣
↪\(downloadRequest.key) Error: \(error)")
            }
        } else {
            print("Error downloading: \(downloadRequest.
↪key) Error: \(error)")
        }
        return nil
    }
    print("Download complete for: \(downloadRequest.key)
↪")
    let downloadOutput = task.result
    return nil
})
```

Objective-C

```
[[transferManager download:downloadRequest ]␣
↪continueWithExecutor:[AWSExecutor mainThreadExecutor]
    withBlock:^id(AWSTask *task) {
    if (task.error){
        if ([task.error.domain isEqualToString:
↪AWSS3TransferManagerErrorDomain]) {
            switch (task.error.code) {
                case AWSS3TransferManagerErrorCancelled:
                case AWSS3TransferManagerErrorPaused:
                break;

                default:
                    NSLog(@"Error: %@", task.error);
                    break;
            }

        } else {
            NSLog(@"Error: %@", task.error);
        }
    }

    if (task.result) {
        AWSS3TransferManagerDownloadOutput␣
↪*downloadOutput = task.result;
    }
    return nil;
}];
```

Upload

Swift

```
transferManager.upload(uploadRequest).
↪continueWith(executor: AWSExecutor.mainThread(), block:
↪ { (task:AWSTask<AnyObject>) -> Any? in

    if let error = task.error as? NSError {
        if error.domain ==␣
↪AWSS3TransferManagerErrorDomain, let code =␣
↪AWSS3TransferManagerErrorType(rawValue: error.code) {
            switch code {
            case .cancelled, .paused:
                break
            default:
                print("Error uploading: \(uploadRequest.
↪key) Error: \(error)")
            }
        } else {
            print("Error uploading: \(uploadRequest.key)␣
↪Error: \(error)")
        }
        return nil
    }
```

```
    let uploadOutput = task.result
    print("Upload complete for: \(uploadRequest.key)")
    return nil
})
```

Objective-C

```
[[transferManager upload:uploadRequest]␣
↪continueWithExecutor:[AWSExecutor mainThreadExecutor]
            withBlock:^id(AWSTask *task) {
if (task.error) {
    if ([task.error.domain isEqualToString:
↪AWSS3TransferManagerErrorDomain]) {
        switch (task.error.code) {
            case AWSS3TransferManagerErrorCancelled:
            case AWSS3TransferManagerErrorPaused:
                break;

            default:
                NSLog(@"Error: %@", task.error);
                break;
        }
    } else {
        // Unknown error.
        NSLog(@"Error: %@", task.error);
    }
}

if (task.result) {
    AWSS3TransferManagerUploadOutput *uploadOutput =␣
↪task.result;
    // The file uploaded successfully.
}
return nil;
```

}];

3. Displaying a Downloaded Image in an UIImageView

The use of *download:* in this example is executed on the main thread. The following code illustrates displaying such an image in a *UIImageView* configured in your project .

Note that it can only succeed after download of the file it displays has completed.

Swift

```
self.imageView.image = UIImage(contentsOfFile:␣
↪downloadingFileURL.path)
```

Objective-C

```
self.imageView.image = [UIImage imageWithContentsOfFile:
↪downloadingFilePath];
```

3.2.3 Pause, Resume, and Cancel Object Transfers

In this section:

The `TransferManager` supports pause, resume, and cancel operations for both uploads and downloads. The *pause*, *cancel*, *resumeAll*, *cancelAll*, *pauseAll*, *upload:*, and *download:* operations all return instances of *AWSTask*. Use these methods with a *continueWith block:* to handle the returns of these operations.

Use continueWith Block to Handle Results

The following code illustrates using *continueWith block:* when calling the *pause* method.

Swift

```
uploadRequest.pause().continueWith(block: { (task:AWSTask
↪<AnyObject>) -> Any? in
    if let error = task.error as? NSError {
        print("Error: \(error)")
        return nil
    }

    // Upload has been paused.
    return nil
})
```

Objective-C

```
[[self.uploadRequest pause] continueWithBlock:^id(AWSTask *task)
↪{
    if (task.error) {
        NSLog(@"Error: %@",task.error);
    } else {
```

```
    }

    // Upload has been paused.
    return nil;
}];
```

For brevity, the following examples omit the *continueWithBlock*.

Pause a Transfer

To pause an object transfer, call *pause* on the request object.

Swift

```
uploadRequest.pause()
downloadRequest.pause()
```

Objective-C

```
[uploadRequest pause];
[downloadRequest pause];
```

Resume a Transfer

To resume a transfer, call *upload* or *download* and pass in the paused request object.

Swift

```
transferManager.upload(uploadRequest)
transferManager.download(downloadRequest)
```

Objective-C

```
[transferManager upload:uploadRequest];
[transferManager download:downloadRequest];
```

Cancel a Transfer

To cancel a transfer, call *cancel* on the upload or download request.

Swift

```
uploadRequest.cancel()
downloadRequest.cancel()
```

Objective-C

```
[uploadRequest cancel];
[downloadRequest cancel];
```

Pause All Transfers

To pause all of the current upload and download requests, call *pauseAll* on the `TransferManager`.

Swift

```
transferManager.pauseAll()
```

Objective-C

```
[transferManager pauseAll];
```

Resume All Transfers

To resume all of the current upload and download requests, call *resumeAll* on the `TransferManager` passing an *AWSS3''TransferManager''ResumeAllBlock*, which is a closure that takes *AWSRequest* as a parameter, and can be used to reset the progress blocks for the requests.

Swift

```
transferManager.resumeAll({ (request:AWSRequest?) in
    // All paused requests have resumed.
})
```

Objective-C

```
[transferManager resumeAll:^(AWSRequest *request) {
    // All paused requests have resumed.
}];
```

Cancel All Transfers

To cancel all upload and download requests, call *cancelAll* on the TransferManager.

Swift

```
transferManager.cancelAll()
```

Objective-C

```
[transferManager cancelAll];
```

3.2.4 Track Progress

Using the *uploadProgress* and *downloadProgress* blocks, you can track the progress of object transfers. These blocks work in conjunction with the Grand Central Dispatch *dispatch_async* function, as shown in the following examples.

Track the progress of an upload.

Swift

```
uploadRequest.uploadProgress = {(bytesSent: Int64,␣
↪totalBytesSent: Int64, totalBytesExpectedToSend:␣
↪Int64) -> Void in
    DispatchQueue.main.async(execute: {() -> Void in
        //Update progress
    })
}
```

Objective-C

```
uploadRequest.uploadProgress =  ^(int64_t bytesSent,␣
↪int64_t totalBytesSent, int64_t␣
↪totalBytesExpectedToSend){
    dispatch_async(dispatch_get_main_queue(), ^{
     //Update progress
});
```

Track the progress of a download.

Swift

```
downloadRequest.downloadProgress = {(bytesSent: Int64,␣
↪totalBytesSent: Int64, totalBytesExpectedToSend:␣
↪Int64) -> Void in
    DispatchQueue.main.async(execute: {() -> Void in
        //Update progress
    })
}
```

Objective-C

```
downloadRequest.downloadProgress = ^(int64_t␣
↪bytesWritten, int64_t totalBytesWritten, int64_t␣
↪totalBytesExpectedToWrite){
dispatch_async(dispatch_get_main_queue(), ^{
        //Update progress
});
```

3.2.5 Multipart Upload

Amazon S3 provides a multipart upload feature to upload a single object as a set of parts. Each part is a contiguous portion of the object's data. The object parts are uploaded independently and in any order. If transmission of any part fails, you can retransmit that part without affecting other parts. After all parts of the object are uploaded, Amazon S3 assembles these parts and creates the object.

In the AWS Mobile SDK for iOS, the `TransferManager` handles multipart upload for you. The minimum part size for a multipart upload is 5MB.

3.2.6 Additional Resources

- Amazon Simple Storage Service Getting Started Guide
- Amazon Simple Storage Service API Reference
- Amazon Simple Storage Service Developer Guide

3.3 Amazon S3 TransferUtility for iOS

On this page:

Amazon Simple Storage Service (S3)

Amazon Simple Storage Service (S3) provides secure, durable, highly scalable object storage in the cloud. Using the AWS Mobile SDK for iOS, you can directly access Amazon S3 from your mobile app. For information about Amazon S3 regional availability, see AWS Service Region Availability.

TransferUtility Features

In addition to downloading, uploading, pausing, resuming and cancelling transfers, use the Amazon S3 TransferUtility class to transfer data to a file without saving a source file. This class also completes transfers in the background, even if the system suspends your app. User choice to suspend an app cancels in-progress transfers.

Should I Use `TransferManager` or `TransferUtility`?

To choose which API best suits your needs, see *Should I Use TransferManager or TransferUtility?*

3.3.1 Setup

To set your project up to use `TransferUtility`, take the steps below.

1. Setup the SDK, Credentials, and Services

Follow the steps described in *Amazon S3 Integration Setup* to install the AWS Mobile SDK for iOS and configure AWS services, credentials, and permissions.

2. Import the SDK Amazon S3 APIs

Add the following import statements to your Xcode project.

Swift

```
import AWSS3
```

Objective-C

```
#import <AWSS3/AWSS3.h>
```

3. Configure the Application Delegate

The Transfer Utility for iOS uses the background transfer feature in iOS to continue data transfers even when your app isn't running.

Call the following method in `-application:handleEventsForBackgroundURLSession: completionHandler:` of your application delegate. When the app in the foreground, the delegate enables iOS to notify `TransferUtility` that a transfer has completed.

Swift

```
func application(_ application: UIApplication,␣
↪handleEventsForBackgroundURLSession identifier:␣
↪String, completionHandler: @escaping () -> Void) {
    // Store the completion handler.
    AWSS3TransferUtility.
↪interceptApplication(application,␣
↪handleEventsForBackgroundURLSession: identifier,␣
↪completionHandler: completionHandler)
}
```

Objective-C

```
- (void)application:(UIApplication *)application␣
↪handleEventsForBackgroundURLSession:(NSString␣
↪*)identifier
completionHandler:(void (^)())completionHandler {
    /* Store the completion handler.*/
    [AWSS3TransferUtility interceptApplication:
↪application handleEventsForBackgroundURLSession:
↪identifier completionHandler:completionHandler];
}
```

3.3.2 Uploading a File

The following code for uploading a file by calling `uploadFile:` on `AWSS3TransferUtility` uses the pattern that is common to all the types of transfers `TransferUtility` supports. For code examples for other kinds of transfer, see *More Transfer Examples*. .

Swift

```
let fileURL = // The file to upload
let  transferUtility = AWSS3TransferUtility.default()
transferUtility.uploadFile(fileURL,
        bucket: S3BucketName,
        key: S3UploadKeyName,
        contentType: "image/png",
        expression: nil,
        completionHandler: nil).continueWith {
    (task) -> AnyObject! in if let error = task.error {
        print("Error: \(error.localizedDescription)")
    }

    if let _ = task.result {
        // Do something with uploadTask.
    }
    return nil;
}
```

Objective-C

```
NSURL *fileURL = // The file to upload.

AWSS3TransferUtility *transferUtility = [AWSS3TransferUtility␣
↪defaultS3TransferUtility];
[[transferUtility uploadFile:fileURL
                      bucket:@"YourBucketName"
                      key:@"YourObjectKeyName"
                      contentType:@"text/plain"
                      expression:nil
            completionHander:nil] continueWithBlock:^id(AWSTask␣
↪*task) {
    if (task.error) {
        NSLog(@"Error: %@", task.error);
    }
    if (task.result) {
        AWSS3TransferUtilityUploadTask *uploadTask = task.result;
        // Do something with uploadTask.
    }

    return nil;
}];
```

3.3.3 Tracking Progress and Completion

Implement progress and completion actions for TransferManager transfers by passing *progressBlock* and *completionHandler* blocks to the call to `TransferUtility` that initiates the transfer.

The following example of initiating a data upload shows how progress and completion handling is typically done for all transfers.

Swift

```
let data = // The data to upload

let expression = AWSS3TransferUtilityUploadExpression()
expression.progressBlock = {(task, progress) in DispatchQueue.
↪main.async(execute: {
        // Do something e.g. Update a progress bar.
    })
}

let completionHandler = { (task, error) -> Void in
    DispatchQueue.main.async(execute: {
        // Do something e.g. Alert a user for transfer␣
↪completion.
        // On failed uploads, `error` contains the error object.
    })
}

let  transferUtility = AWSS3TransferUtility.default()

transferUtility.uploadData(data,
            bucket: S3BucketName,
            key: S3UploadKeyName,
            contentType: "image/png",
            expression: expression,
            completionHandler: completionHandler).continueWith {␣
↪(task) -> AnyObject! in
    if let error = task.error {
        print("Error: \(error.localizedDescription)")
    }

    if let _ = task.result {
        // Do something with uploadTask.
    }

    return nil;
}
```

Objective-C

```
NSData *dataToUpload = // The data to upload.

AWSS3TransferUtilityUploadExpression *expression =␣
↪[AWSS3TransferUtilityUploadExpression new];
expression.progressBlock = ^(AWSS3TransferUtilityTask *task,␣
↪NSProgress *progress) {
```

```
    dispatch_async(dispatch_get_main_queue(), ^{
        // Do something e.g. Update a progress bar.
    });
};


AWSS3TransferUtilityUploadCompletionHandlerBlock␣
↪completionHandler = ^(AWSS3TransferUtilityUploadTask *task,␣
↪NSError *error) {
    dispatch_async(dispatch_get_main_queue(), ^{
        // Do something e.g. Alert a user for transfer␣
↪completion.
        // On failed uploads, `error` contains the error object.
    });
};

AWSS3TransferUtility *transferUtility = [AWSS3TransferUtility␣
↪defaultS3TransferUtility];
[[transferUtility uploadData:dataToUpload
                      bucket:@"YourBucketName"
                      key:@"YourObjectKeyName"
                      contentType:@"text/plain"
                      expression:expression
            completionHander:completionHandler] continueWithBlock:^
↪id(AWSTask *task) {
    if (task.error) {
        NSLog(@"Error: %@", task.error);
    }
    if (task.result) {
       AWSS3TransferUtilityUploadTask *uploadTask = task.result;
        // Do something with uploadTask.
    }

    return nil;
}];
```

3.3.4 Managing Transfers

This section describes how to manage `TransferUtility` transfers at different points in the app lifecycle.

With the App in the Foreground

To suspend, resume, and cancel uploads and downloads, retain references to `AWSS3TransferUtilityUploadTask` and `AWSS3TransferUtilityDownloadTask`. To manage data transfers call `suspend`, `resume`, and `cancel` on those tasks. The following example shows the `cancel` method being called on an upload.

Swift

```
transferUtility.uploadFile(fileURL,
        bucket: S3BucketName,
        key: S3UploadKeyName,
        contentType: "image/png",
        expression: nil,
        completionHandler: nil).continueWith {
    (task) -> AnyObject! in if let error = task.error {
        print("Error: \(error.localizedDescription)")
    }

    if let uploadTask = task.result {
        uploadTask.cancel()
    }
    return nil;
}
```

Objective-C

```
[[transferUtility uploadFile:fileURL
                      bucket:@"YourBucketName"
                      key:@"YourObjectKeyName"
                      contentType:@"text/plain"
                      expression:nil
            completionHander:nil] continueWithBlock:^id(AWSTask␣
↪*task) {
    if (task.error) {
        NSLog(@"Error: %@", task.error);
    }
    if (task.result) {
        AWSS3TransferUtilityUploadTask *uploadTask = task.result;
        [uploadTask cancel]
    }

    return nil;
}];
```

When a Suspended App Returns to the Foreground

Upon App Returning to the Foreground

When an app that has initiated a `TransferUtility` transfer becomes suspended and then returns to the foreground, the transfer may still be in progress or may have completed. In both cases, use the following code to reestablish the transfer as being handled by progress and completion blocks of the app in the foreground.

The code uses downloading a file as the example but the pattern also works for upload:

You receive `AWSS3TransferUtilityUploadTask` and `AWSS3TransferUtilityDownloadTask` when you initiate the upload and download respectively. These tasks have a property called `taskIdentifier`, which uniquely identifies the transfer task object within the Transfer Utility. Your app should persist the identifier through closure and relaunch, so that you

can uniquely identify the task objects when the app is comes back into the foreground.

Swift

```
override func viewDidLoad() {
super.viewDidLoad()

...

let transferUtility = AWSS3TransferUtility.default()

var uploadProgressBlock: AWSS3TransferUtilityProgressBlock? =
↪{(task: AWSS3TransferUtilityTask, progress: Progress) in
    DispatchQueue.main.async {
        // Handle progress feedback, e.g. update progress bar
    }
}
var downloadProgressBlock: AWSS3TransferUtilityProgressBlock? = {
    (task: AWSS3TransferUtilityTask, progress: Progress) in␣
↪DispatchQueue.main.async {
        // Handle progress feedback, e.g. update progress bar
    }
}
var completionBlockUpload:
↪AWSS3TransferUtilityUploadCompletionHandlerBlock? = {
    (task, error) in DispatchQueue.main.async {
        // perform some action on completed upload operation
    }
}
var completionBlockDownload:
↪AWSS3TransferUtilityDownloadCompletionHandlerBlock? = {
    (task, url, data, error) in DispatchQueue.main.async {
        // perform some action on completed download operation
    }
}

transferUtility.enumerateToAssignBlocks(forUploadTask: {
    (task, progress, completion) -> Void in

        let progressPointer = AutoreleasingUnsafeMutablePointer
↪<AWSS3TransferUtilityProgressBlock?>(& uploadProgressBlock)

        let completionPointer = AutoreleasingUnsafeMutablePointer
↪<AWSS3TransferUtilityUploadCompletionHandlerBlock?>(&
↪completionBlockUpload)

        // Reassign your progress feedback
        progress?.pointee = progressPointer.pointee

        // Reassign your completion handler.
        completion?.pointee = completionPointer.pointee

}, downloadTask: {
    (task, progress, completion) -> Void in
```

```
        let progressPointer = AutoreleasingUnsafeMutablePointer
↪<AWSS3TransferUtilityProgressBlock?>(&downloadProgressBlock)

        let completionPointer = AutoreleasingUnsafeMutablePointer
↪<AWSS3TransferUtilityDownloadCompletionHandlerBlock?>(&
↪completionBlockDownload)

        // Reassign your progress feedback
        progress?.pointee = progressPointer.pointee

        // Reassign your completion handler.
        completion?.pointee = completionPointer.pointee
})

 if let downloadTask = task.result {
    // Do something with downloadTask.
}
```

Objective-C

```
- (void)viewDidLoad {
    [super viewDidLoad];

    ...

    AWSS3TransferUtility *transferUtility =␣
↪[AWSS3TransferUtility defaultS3TransferUtility];
    [transferUtility enumerateToAssignBlocksForUploadTask:^(
        AWSS3TransferUtilityUploadTask *uploadTask,
        __autoreleasing AWSS3TransferUtilityUploadProgressBlock␣
↪*uploadProgressBlockReference,
        __autoreleasing␣
↪AWSS3TransferUtilityUploadCompletionHandlerBlock␣
↪*completionHandlerReference
    ) {
        NSLog(@"%lu", (unsigned long)uploadTask.taskIdentifier);

        // Use `uploadTask.taskIdentifier` to determine what␣
↪blocks to assign.

        *uploadProgressBlockReference = ...; // Reassign your␣
↪progress feedback block.
        *completionHandlerReference = ...; // Reassign your␣
↪completion handler.
    }
    downloadTask:^(AWSS3TransferUtilityDownloadTask␣
↪*downloadTask, __autoreleasing␣
↪AWSS3TransferUtilityDownloadProgressBlock␣
↪*downloadProgressBlockReference, __autoreleasing␣
↪AWSS3TransferUtilityDownloadCompletionHandlerBlock␣
↪*completionHandlerReference) {
        NSLog(@"%lu", (unsigned long)downloadTask.
↪taskIdentifier);
```

```
            // Use `downloadTask.taskIdentifier` to determine␣
↪what blocks to assign.
        *downloadProgressBlockReference =  // Reassign your␣
↪progress feedback block.
        *completionHandlerReference = // Reassign your completion␣
↪handler.
    }];
}

if (task.result) {
    AWSS3TransferUtilityUploadTask *downloadTask = task.result;
    // Do something with downloadTask.
}
```

3.3.5 More Transfer Examples

This section provides descriptions and abbreviated examples of the aspects of each type of transfer that are unique. For information about typical code surrounding the following snippets see *Managing Transfers* and *Tracking Progress and Completion*.

Downloading to a File

The following code shows how to download a file.

Swift

```
let fileURL = // The file URL of the download destination.

// Add progress and completion blocks
. . .

let  transferUtility = AWSS3TransferUtility.default()
transferUtility.download(
        to: fileURL
        bucket: S3BucketName,
        key: S3DownloadKeyName,
        expression: expression,
        completionHander: completionHandler
).continueWith {
    (task) -> AnyObject! in if let error = task.error {
        print("Error: \(error.localizedDescription)")
    }

    if let _ = task.result {
        // Do something with downloadTask.
    }
    return nil;
}
```

Objective-C

```
NSURL *fileURL = ...; // The file URL of the download␣
↪destination.

// Add progress and completion blocks
. . .

AWSS3TransferUtility *transferUtility = [AWSS3TransferUtility␣
↪defaultS3TransferUtility];
[[transferUtility downloadToURL:nil
                         bucket:S3BucketName
                         key:S3DownloadKeyName
                         expression:expression
               completionHander:completionHandler] continueWithBlock:^
↪id(AWSTask *task) {
    if (task.error) {
        NSLog(@"Error: %@", task.error);
    }
    if (task.result) {
        AWSS3TransferUtilityDownloadTask *downloadTask = task.
↪result;
        // Do something with downloadTask.
    }

    return nil;
}];
```

Uploading Binary Data to a File

To upload data to a file in Amazon S3 call `uploadData:`.

This method saves the data as a file in a temporary directory. The next time `AWSS3TransferUtility` is initialized, the expired temporary files are cleaned up. If you upload many large objects to an Amazon S3 bucket in a short period of time, it is more efficient to use the upload file method and then manually purge the unnecessary temporary files as early as possible.

Swift

```
let data = // The data to upload

// Add progress and completion blocks
. . .

let  transferUtility = AWSS3TransferUtility.default()

transferUtility.uploadData(data,
            bucket: S3BucketName,
            key: S3UploadKeyName,
            contentType: "image/png",
            expression: expression,
            completionHandler: completionHandler).continueWith {␣
↪(task) -> AnyObject! in
```

```
    if let error = task.error {
        print("Error: \(error.localizedDescription)")
    }

    if let _ = task.result {
        // Do something with uploadTask.
    }

    return nil;
}
```

Objective-C

```
NSData *dataToUpload = // The data to upload.

// Add progress and completion blocks
. . .

AWSS3TransferUtility *transferUtility = [AWSS3TransferUtility␣
↪defaultS3TransferUtility];
[[transferUtility uploadData:dataToUpload
                      bucket:@"YourBucketName"
                      key:@"YourObjectKeyName"
                      contentType:@"text/plain"
                      expression:expression
            completionHander:completionHandler] continueWithBlock:^
↪id(AWSTask *task) {
    if (task.error) {
        NSLog(@"Error: %@", task.error);
    }
    if (task.result) {
        AWSS3TransferUtilityUploadTask *uploadTask = task.result;
        // Do something with uploadTask.
    }

    return nil;
}];
```

Downloading Binary Data to a File

The following code shows how to download binary a file.

Swift

```
let fileURL = // The file URL of the download destination.

// Add progress and completion blocks
. . .

let  transferUtility = AWSS3TransferUtility.default()
transferUtility.downloadData(
        fromBucket: S3BucketName,
```

```
        key: S3DownloadKeyName,
        expression: expression,
        completionHander: completionHandler
).continueWith {
    (task) -> AnyObject! in if let error = task.error {
        print("Error: \(error.localizedDescription)")
    }

    if let _ = task.result {
        // Do something with downloadTask.
    }

    return nil;
}
```

Objective-C

```
AWSS3TransferUtilityDownloadExpression *expression =␣
↪[AWSS3TransferUtilityDownloadExpression new];

// Add progress and completion blocks
. . .

AWSS3TransferUtility *transferUtility = [AWSS3TransferUtility␣
↪defaultS3TransferUtility];
[[transferUtility downloadDataFromBucket:S3BucketName
    key:S3DownloadKeyName
    expression:expression
    completionHander:completionHandler] continueWithBlock:^
↪id(AWSTask *task) {
        if (task.error) {
            NSLog(@"Error: %@", task.error);
        }
        if (task.result) {
            AWSS3TransferUtilityDownloadTask *downloadTask =␣
↪task.result;
            // Do something with downloadTask.
        }

        return nil;
    }
];
```

3.3.6 Limitations

The S3 Transfer Utility generates Amazon S3 pre-signed URLs to use for background data transfer. Using Amazon Cognito Identity, you receive AWS temporary credentials. The credentials are valid for up to 60 minutes. At the same time, generated S3 pre-signed URLs cannot last longer than that time. Because of this limitation, the Amazon S3 Transfer Utility enforces 50 minute transfer timeouts, leaving a 10 minute buffer before AWS temporary credentials are regenerated. After 50 minutes, you receive a transfer failure.

If you need to transfer data that cannot be transferred in under 50 minutes, use *AWSS3* instead.

3.4 Amazon S3 Pre-Signed URLs: For Background Transfer

If you are working with large file transfers, you may want to perform uploads and downloads in the background. To do this, you need to create a background session using `NSURLSession` and then transfer your objects using pre-signed URLs.

The following sections describe pre-signed S3 URLs. To learn more about `NSURLSession`, see Using NSURLSession.

3.4.1 Pre-Signed URLs

By default, all Amazon S3 resources are private. If you want your users to have access to Amazon S3 buckets or objects, you can assign appropriate permissions with an IAM policy.

Alternatively, you can use pre-signed URLs to give your users access to Amazon S3 objects. A pre-signed URL provides access to an object without requiring AWS security credentials or permissions.

When you create a pre-signed URL, you must provide your security credentials, specify a bucket name, an object key, an HTTP method, and an expiration date and time. The pre-signed URL is valid only for the specified duration.

3.4.2 Build a Pre-Signed URL

The following example shows how to build a pre-signed URL for an Amazon S3 download in the background.

Swift

```
AWSS3PreSignedURLBuilder.default().
↪getPreSignedURL(getPreSignedURLRequest).continueWith { (task:
↪AWSTask<NSURL>) -> Any? in
    if let error = task.error as? NSError {
        print("Error: \(error)")
        return nil
    }

    let presignedURL = task.result
    print("Download presignedURL is: \(presignedURL)")

    let request = URLRequest(url: presignedURL as! URL)
    let downloadTask: URLSessionDownloadTask = URLSession.shared.
↪downloadTask(with: request)
    downloadTask.resume()

    return nil
}
```

Objective-C

```
AWSS3GetPreSignedURLRequest *getPreSignedURLRequest =␣
↪[AWSS3GetPreSignedURLRequest new];
getPreSignedURLRequest.bucket = @"myBucket";
getPreSignedURLRequest.key = @"myImage.jpg";
getPreSignedURLRequest.HTTPMethod = AWSHTTPMethodGET;
getPreSignedURLRequest.expires = [NSDate␣
↪dateWithTimeIntervalSinceNow:3600];

[[[AWSS3PreSignedURLBuilder defaultS3PreSignedURLBuilder]␣
↪getPreSignedURL:getPreSignedURLRequest]
    continueWithBlock:^id(AWSTask *task) {

    if (task.error) {
        NSLog(@"Error: %@",task.error);
    } else {

        NSURL *presignedURL = task.result;
        NSLog(@"download presignedURL is: \n%@", presignedURL);

        NSURLRequest *request = [NSURLRequest requestWithURL:
↪presignedURL];
        self.downloadTask = [self.session␣
↪downloadTaskWithRequest:request];
        //downloadTask is an instance of␣
↪NSURLSessionDownloadTask.
        //session is an instance of NSURLSession.
        [self.downloadTask resume];

    }
    return nil;
}];
```

The [receding example uses `GET` as the HTTP method: `AWSHTTPMethodGET`. For an upload request to Amazon S3, we would need to use a PUT method and also specify a content type.

Swift

```
getPreSignedURLRequest.httpMethod = .PUT
let fileContentTypeStr = "text/plain"
getPreSignedURLRequest.contentType = fileContentTypeStr
```

Objective-C

```
getPreSignedURLRequest.HTTPMethod = AWSHTTPMethodPUT;
NSString *fileContentTypeStr = @"text/plain";
getPreSignedURLRequest.contentType = fileContentTypeStr;
```

Here's an example of building a pre-signed URL for a background upload to S3.

Swift

```
let getPreSignedURLRequest = AWSS3GetPreSignedURLRequest()
getPreSignedURLRequest.bucket = "myBucket"
getPreSignedURLRequest.key = "myFile.txt"
getPreSignedURLRequest.httpMethod = .PUT
getPreSignedURLRequest.expires = Date(timeIntervalSinceNow: 3600)

//Important: set contentType for a PUT request.
let fileContentTypeStr = "text/plain"
getPreSignedURLRequest.contentType = fileContentTypeStr

AWSS3PreSignedURLBuilder.default().
↪getPreSignedURL(getPreSignedURLRequest).continueWith { (task:
↪AWSTask<NSURL>) -> Any? in
    if let error = task.error as? NSError {
        print("Error: \(error)")
        return nil
    }

    let presignedURL = task.result
    print("Download presignedURL is: \(presignedURL)")

    var request = URLRequest(url: presignedURL as! URL)
    request.cachePolicy = .reloadIgnoringLocalCacheData
    request.httpMethod = "PUT"
    request.setValue(fileContentTypeStr, forHTTPHeaderField:
↪"Content-Type")

    let uploadTask: URLSessionTask = URLSession.shared.
↪uploadTask(with: request, fromFile: URL(fileURLWithPath: "your/
↪file/path/myFile.txt"))
    uploadTask.resume()

    return nil
}
```

Objective-C

```
AWSS3GetPreSignedURLRequest *getPreSignedURLRequest =␣
↪[AWSS3GetPreSignedURLRequest new];
getPreSignedURLRequest.bucket = @"myBucket";
getPreSignedURLRequest.key = @"myFile";
getPreSignedURLRequest.HTTPMethod = AWSHTTPMethodPUT;
getPreSignedURLRequest.expires = [NSDate␣
↪dateWithTimeIntervalSinceNow:3600];

//Important: set contentType for a PUT request.
NSString *fileContentTypeStr = @"text/plain";
getPreSignedURLRequest.contentType = fileContentTypeStr;

[[[AWSS3PreSignedURLBuilder defaultS3PreSignedURLBuilder]␣
↪getPreSignedURL:getPreSignedURLRequest]
                                                    ␣
↪continueWithBlock:^id(AWSTask *task) {
```

```
    if (task.error) {
        NSLog(@"Error: %@",task.error);
    } else {
        NSURL *presignedURL = task.result;
        NSLog(@"upload presignedURL is: \n%@", presignedURL);

        NSMutableURLRequest *request = [NSMutableURLRequest␣
↪requestWithURL:presignedURL];
        request.cachePolicy =␣
↪NSURLRequestReloadIgnoringLocalCacheData;
        [request setHTTPMethod:@"PUT"];
        [request setValue:fileContentTypeStr forHTTPHeaderField:@
↪"Content-Type"];

        self.uploadTask = [self.session uploadTaskWithRequest:
↪request fromFile:self.uploadFileURL];
        //uploadTask is an instance of NSURLSessionDownloadTask.
        //session is an instance of NSURLSession.
        [self.uploadTask resume];
    }
    return nil;
}];
```

Additional Resources

- Amazon Simple Storage Service Getting Started Guide
- Amazon Simple Storage Service API Reference
- Amazon Simple Storage Service Developer Guide

3.5 Amazon S3 Server-Side Encryption Support in iOS

The AWS Mobile SDK for iOS supports server-side encryption of Amazon S3 data. To learn more about server-side encryption, see PUT Object.

The following properties are available to configure the encryption:

- SSECustomerAlgorithm
- SSECustomerKey
- SSECustomerKeyMD5
- AWSS3ServerSideEncryption

To use these properties, import the `AWSSS3Model` with the following statement.

Swift

```
import AWSS3
```

Objective-C

```
#import <AWSS3/AWSS3.h>
```

`SSECustomerAlgorithm` is a property of `AWSS3ReplicateObjectOutput`. If server-side encryption with a customer-provided encryption key was requested, the response will include this header, which confirms the encryption algorithm that was used. Currently, the only valid option is AES256. You can access `SSECustomerAlgorithm` as follows.

Swift

```
let replicateObjectOutput = AWSS3ReplicateObjectOutput()
replicateObjectOutput?.sseCustomerAlgorithm =
↪"mySseCustomerAlgorithm
```

Objective-C

```
AWSS3ReplicateObjectOutput *replicateObjectOutput =␣
↪[AWSS3ReplicateObjectOutput new];
replicateObjectOutput.SSECustomerAlgorithm = @
↪"mySseCustomerAlgorithm";
```

`SSECustomerKey`, a property of `AWSS3UploadPartRequest`, specifies the customer-provided encryption key for Amazon S3 to use to encrypting data. This value is used to store the object, and is then discarded; Amazon doesn't store the encryption key. The key must be appropriate for use with the algorithm specified in the `x-amz-server-side-encryption-customer-algorithm` header. This must be the same encryption key specified in the request to initiate a multipart upload. You can access SSECustomerKey as follows.

Swift

```
let uploadPartRequest = AWSS3UploadPartRequest()
uploadPartRequest?.sseCustomerKey =
↪"customerProvidedEncryptionKey"
```

Objective-C

```
AWSS3UploadPartRequest *uploadPartRequest =␣
↪[AWSS3UploadPartRequest new];
uploadPartRequest.SSECustomerKey = @
↪"customerProvidedEncryptionKey";
```

`SSECustomerKeyMD5` is a property of `AWSS3PutObjectOutput`. If server-side encryption with a customer-provided encryption key is requested, the response will include this header. The response provides round trip message integrity verification of the customer-provided encryption key. You can access `SSECustomerKeyMD5` as follows.

Swift

```
let objectOutput = AWSS3PutObjectOutput()
// Access objectOutput?.sseCustomerKeyMD5 ...
```

Objective-C

```
AWSS3PutObjectOutput *objectOutput = [AWSS3PutObjectOutput new];
//Access objectOutput.SSECustomerKeyMD5 ...
```

`AWSS3ServerSideEncryption` represents the encryption algorithm for storing an object in Amazon S3. You can access it as follows.

Swift

```
let objectOutput = AWSS3PutObjectOutput()
// Access objectOutput?.sseCustomerKeyMD5 ...
```

Objective-C

```
AWSS3ReplicateObjectOutput *replicateObjectOutput =␣
↪[AWSS3ReplicateObjectOutput new];
// Access replicateObjectOutput.serverSideEncryption ...
```

3.5.1 Additional Resources

- Amazon Simple Storage Service Getting Started Guide
- Amazon Simple Storage Service API Reference
- Amazon Simple Storage Service Developer Guide

3.6 Should I Use TransferManager or TransferUtility?

Both the Amazon S3 TransferManager and TransferUtitliy classes make it easy for you to upload and download files from Amazon S3 while optimizing for performance and reliability. Both hide the complexity of making asynchronous file transfers behind simple APIs. Both provide the ability to pause, resume, and cancel file transfers.

The differences are as follows.

- Use `TransferUtility` to:
  - Make background file or data transfers that complete even if the system suspends an app invoking the transfer
  - Transfer binary data to a file with out saving a file at the transfer source location first.
- Use `TransferManager` to make file transfers that happen while the app is in the foreground.

  Whenever possible, `TransferManager` uploads are broken up into multiple pieces. Several pieces can be sent in parallel to provide better throughput that is resilient to I/O errors.

3.7 Additional Resources

For information about Amazon S3 regional availability, see AWS Service Region Availability.

- Amazon Simple Storage Service Getting Started Guide
- Amazon Simple Storage Service API Reference
- Amazon Simple Storage Service Developer Guide

CHAPTER 4

Amazon DynamoDB: Store and Retrieve Data

In this section:

4.1 Amazon DynamoDB Integration Setup

This section provides a step-by-step guide for getting started with Amazon DynamoDB using the AWS Mobile SDK for iOS.

4.1.1 Get the SDK

To use Amazon DynamoDB in your mobile app, first set up the AWS Mobile SDK for iOS:

1. Download the iOS SDK and include it in your iOS project, as described in *Set Up the SDK for iOS*.
2. Install the SDK

   Add the AWS SDK for iOS to your project and import the APIs you need, by following the steps in *Set Up the SDK for iOS*.
3. Configure credentials

   To use Amazon Cognito to create AWS identities and credentials that give your users access to your app's AWS resources, follow the steps in *Amazon Cognito for iOS*.

For more information on setting up the Amazon Cognito client, see Using Federated Identities in the *Amazon Cognito Developer Guide*.

4.1.2 Create an Amazon DynamoDB Table and Index

This tutorial is based on a simple bookstore app. The app tracks the books that are available in the bookstore using an Amazon DynamoDB table.

To create the Books table:

1. Sign in to the Amazon DynamoDB Console.
2. Choose *Create Table*.

3. Type **Books** as the name of the table.
4. Enter **ISBN** in the *Partition key* field of the *Primary key* with *String* as their type.
5. Clear the *Use default settings* checkbox and choose + *Add Index*.
6. In the *Add Index* dialog type **Author** with *String* as the type.
7. Check the *Add sort key* checkbox and enter **Title** as the sort key value, with *String* as its type.
8. Leave the other values at their defaults. Choose *Add index* to add the **Author-Title-index** index.
9. Set the read capacity to *10* and the write capacity to *5*.
10. Choose *Create*.Amazon DynamoDB will create your database.
11. Refresh the console and choose your Books table from the list of tables.
12. Open the *Overview* tab and copy or note the Amazon Resource Name (ARN). You need this for the next procedure.

4.1.3 Set Permissions

To use Amazon DynamoDB in your mobile app, you must set the correct permissions. The following IAM policy allows the user to perform the actions shown in this tutorial on two resources (a table and an index) identified by an ARN.

```
{
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "dynamodb:DeleteItem",
            "dynamodb:GetItem",
            "dynamodb:PutItem",
            "dynamodb:Scan",
            "dynamodb:Query",
            "dynamodb:UpdateItem",
            "dynamodb:BatchWriteItem"
        ],
        "Resource": [
            "arn:aws:dynamodb:us-west-2:123456789012:table/Books",
            "arn:aws:dynamodb:us-west-2:123456789012:table/Books/
↪index/*"
        ]
    }]
}
```

Apply this policy to the unauthenticated role assigned to your Amazon Cognito identity pool, replacing the *Resource* values with the correct ARN for the Amazon DynamoDB table:

1. Sign in to the IAM console.
2. Choose *Roles* and then choose the "Unauth" role that Amazon Cognito created for you.

3. Choose *Attach Role Policy*.
4. Choose *Custom Policy* and then Choose *Select*.
5. Type a name for your policy and paste in the policy document shown above, replacing the *Resource* values with the ARNs for your table and index. (You can retrieve the table ARN from the *Details* tab of the database; then append `/index/*` to obtain the value for the index ARN.
6. Choose *Apply Policy*.

To learn more about IAM policies, see Using IAM. To learn more about creating fine-grained access policies for Amazon DynamoDB, see DynamoDB on Mobile – Part 5: Fine-Grained Access Control.

4.2 Amazon DynamoDB Object Mapper API

On this page:

Amazon DynamoDB

Amazon DynamoDB is a fast, highly scalable, highly available, cost-effective, non-relational database service. Amazon DynamoDB removes traditional scalability limitations on data storage while maintaining low latency and predictable performance.

The AWS Mobile SDK for iOS provides both low-level and high-level libraries for working with Amazon DynamoDB.

The high-level library described in this section provides Amazon DynamoDB object mapper which lets you map client-side classes to tables. Working within the data model defined on your client you can write simple, readable code that stores and retrieves objects in the cloud.

The *Amazon DynamoDB Low-level Client* provides useful ways to perform operations like conditional writes and batch operations.

4.2.1 Setup

To set your project up to use the AWS SDK for iOS TransferUtility, take the following steps.

1. Setup the SDK, Credentials, and Services

Follow the steps described in *Amazon DynamoDB Integration Setup* to install the AWS Mobile SDK for iOS and configure AWS services, credentials, and permissions.

4.2.2 Instantiating the Object Mapper API

In this section:

Import the AWSDynamoDB APIs

Add the following import statement to your project.

Swift

```
import AWSDynamoDB
```

Objective-C

```
#import <AWSDynamoDB/AWSDynamoDB.h>
```

Create Amazon DynamoDB Object Mapper Client

Use the AWSDynamoDBObjectMapper to map a client-side class to your database. The object mapper supports high-level operations like creating, getting, querying, updating, and deleting records. Create an object mapper as follows.

Swift

```
dynamoDBObjectMapper = AWSDynamoDBObjectMapper.default()
```

Objective-C

```
AWSDynamoDBObjectMapper *dynamoDBObjectMapper =␣
↪[AWSDynamoDBObjectMapper defaultDynamoDBObjectMapper];
```

Object mapper methods return an `AWSTask` object. for more information, see *Using AWSTask*.

Define a Mapping Class

An Amazon DynamoDB database is a collection of tables, and a table can be described as follows:

- A table is a collection of items.
- Each item is a collection of attributes.
- Each attribute has a name and a value.

For the bookstore app, each item in the table represents a book, and each item has four attributes: *Title*, *Author*, *Price*, and *ISBN*.

Each item (Book) in the table has a *Primary key*, in this case, the primary key is `ISBN`.

To directly manipulate database items through their object representation, map each item in the Book table to a `Book` object in the client-side code, as shown in the following code. Attribute names are case sensitive.

Swift

```
import AWSDynamoDB

class Book : AWSDynamoDBObjectModel, AWSDynamoDBModeling  {
    var Title:String?
    var Author:String?
    var Price:String?
    var ISBN:String?

    class Amazon DynamoDBTableName() -> String {
        return "Books"
    }

    class func hashKeyAttribute() -> String {
        return "ISBN"
    }
}
```

Objective-C

```
#import <AWSDynamoDB/AWSDynamoDB.h>
#import "Book.h"

@interface Book : AWSDynamoDBObjectModel <AWSDynamoDBModeling>

@property (nonatomic, strong) NSString *Title;
@property (nonatomic, strong) NSString *Author;
@property (nonatomic, strong) NSNumber *Price;
@property (nonatomic, strong) NSString *ISBN;

@end


@implementation Book

+ (NSString *)dynamoDBTableName {
```

```
    return @"Books";
}

+ (NSString *)hashKeyAttribute {
    return @"ISBN";
}

@end
```

Note: As of SDK version 2.0.16, the `AWSDynamoDBModel` mapping class is deprecated and replaced by `AWSDynamoDBObjectModel`. For information on migrating your legacy code, see awsdynamodb-model.

To conform to the `AWSDynamoDBModeling` protocol, implement `dynamoDBTableName`, which returns the name of the table, and `hashKeyAttribute`, which returns the name of the primary key. If the table has a range key, implement `+ (NSString *)rangeKeyAttribute`.

4.2.3 CRUD Operations

In this section:

The Amazon DynamoDB table, mapping class, and object mapper client enable your app to interact with objects in the cloud.

Save an Item

The save: method saves an object to Amazon DynamoDB, using the default configuration. As a parameter, `save:` takes a an object that inherits from `AWSDynamoDBObjectModel` and conforms to the `AWSDynamoDBModeling` protocol. The properties of this object will be mapped to attributes in Amazon DynamoDB table.

To create the object to be saved take the following steps.

1. Define the object and it's properties to match your table model.

   Swift

   ```
   let myBook = Book()
   myBook?.ISBN = "3456789012"
   ```

```
myBook?.Title = "The Scarlet Letter"
myBook?.Author = "Nathaniel Hawthorne"
myBook?.Price = 899 as NSNumber?
```

Objective-C

```
Book *myBook = [Book new];
myBook.ISBN = @"3456789012";
myBook.Title = @"The Scarlet Letter";
myBook.Author = @"Nathaniel Hawthorne";
myBook.Price = [NSNumber numberWithInt:899];
```

2. Pass the object to the `save:` method.

Swift

```
dynamoDBObjectMapper.save(myBook).continueWith(block: { (task:
↪AWSTask<AnyObject>!) -> Any? in
    if let error = task.error as? NSError {
        print("The request failed. Error: \(error)")
    } else {
        // Do something with task.result or perform other␣
↪operations.
    }
})
```

Objective-C

```
[[dynamoDBObjectMapper save:myBook]
continueWithBlock:^id(AWSTask *task) {
    if (task.error) {
        NSLog(@"The request failed. Error: [%@]", task.
↪error);
    } else {
        //Do something with task.result or perform other␣
↪operations.
    }
    return nil;
}];
```

Save Behavior Options

The AWS Mobile SDK for iOS supports the following save behavior options:

- ‘‘AWSDynamoDBObjectMapperSaveBehaviorUpdate‘

  This option does not affect unmodeled attributes on a save operation. Passing a nil value for the modeled attribute removes the attribute from the corresponding item in Amazon DynamoDB. By default, the object mapper uses this behavior.

- `AWSDynamoDBObjectMapperSaveBehaviorUpdateSkipNullAttributes`

This option is similar to the default update behavior, except that it ignores any null value attribute(s) and does not remove them from an item in Amazon DynamoDB.

- `AWSDynamoDBObjectMapperSaveBehaviorAppendSet`

  This option treats scalar attributes (String, Number, Binary) the same as the `AWSDynamoDBObjectMapperSaveBehaviorUpdateSkipNullAttributes` option. However, for set attributes, this option appends to the existing attribute value instead of overriding it. The caller must ensure that the modeled attribute type matches the existing set type; otherwise, a service exception occurs.

- `AWSDynamoDBObjectMapperSaveBehaviorClobber`

  This option clears and replaces all attributes, including unmodeled ones, on save. Versioned field constraints are be disregarded.

The following code provides an example of setting a default save behavior on the object mapper.

Swift

```
let updateMapperConfig = AWSDynamoDBObjectMapperConfiguration()
updateMapperConfig.saveBehavior = .updateSkipNullAttributes
```

Objective-C

```
AWSDynamoDBObjectMapperConfiguration *updateMapperConfig =␣
↪[AWSDynamoDBObjectMapperConfiguration new];
updateMapperConfig.saveBehavior =␣
↪AWSDynamoDBObjectMapperSaveBehaviorUpdateSkipNullAttributes;
```

Use `updateMapperConfig` as an argument when calling save:configuration:.

Retrieve an Item

Using an object's primary key, in this case, `ISBN`, we can load the corresponding item from the database. The following code returns the Book item with an ISBN of `6543210987`.

Swift

```
dynamoDBObjectMapper.load(Book.self, hashKey: "6543210987"␣
↪rangeKey:nil).continueWith(block: { (task:AWSTask<AnyObject>!
↪) -> Any? in
    if let error = task.error as? NSError {
        print("The request failed. Error: \(error)")
    } else if let resultBook = task.result as? Book {
        // Do something with task.result.
    }
    return nil
})
```

Objective-C

```
[[dynamoDBObjectMapper load:[Book class] hashKey:@"6543210987"␣
↪rangeKey:nil]
continueWithBlock:^id(AWSTask *task) {
    if (task.error) {
        NSLog(@"The request failed. Error: [%@]", task.error);
    } else {
        //Do something with task.result.
    }
    return nil;
}];
```

The object mapper creates a mapping between the `Book` item returned from the database and the `Book` object on the client (here, `resultBook`). Access the title at `resultBook.Title`.

Since the Books database does not have a range key, `nil` was passed to the `rangeKey` parameter.

Update an Item

To update an item in the database, just set new attributes and save the objects. The primary key of an existing item, `myBook.ISBN` in the `Book` object mapper example, cannot be changed. If you save an existing object with a new primary key, a new item with the same attributes and the new primary key are created.

Delete an Item

To delete a table row, use the *remove:* method.

Swift

```
let bookToDelete = Book()
bookToDelete?.ISBN = "4456789012";

dynamoDBObjectMapper.remove(bookToDelete).continueWith(block: {␣
↪(task:AWSTask<AnyObject>!) -> Any? in
    if let error = task.error as? NSError {
        print("The request failed. Error: \(error)")
    } else {
        // Item deleted.
    }
})
```

Objective-C

```
Book *bookToDelete = [Book new];
bookToDelete.ISBN = @"4456789012";

[[dynamoDBObjectMapper remove:bookToDelete]
 continueWithBlock:^id(AWSTask *task) {

    if (task.error) {
```

```
        NSLog(@"The request failed. Error: [%@]", task.error);
    } else {
        //Item deleted.
    }
    return nil;
}];
```

4.2.4 Perform a Scan

A scan operation retrieves in an undetermined order.

The `scan:expression:` method takes two parameters: the class of the resulting object and an instance of `AWSDynamoDBScanExpression`, which provides options for filtering results.

The following example shows how to create an `AWSDynamoDBScanExpression` object, set its `limit` property, and then pass the `Book` class and the expression object to `scan:expression:`.

Swift

```
let scanExpression = AWSDynamoDBScanExpression()
scanExpression.limit = 20

dynamoDBObjectMapper.scan(Book.self, expression: scanExpression).
↪continueWith(block: { (task:AWSTask<AnyObject>!) -> Any? in
    if let error = task.error as? NSError {
        print("The request failed. Error: \(error)")
    } else if let paginatedOutput = task.result {
        for book in paginatedOutput.items as! Book {
            // Do something with book.
        }
    }
})
```

Objective-C

```
AWSDynamoDBScanExpression *scanExpression =␣
↪[AWSDynamoDBScanExpression new];
scanExpression.limit = @10;

[[dynamoDBObjectMapper scan:[Book class]
        expression:scanExpression]
continueWithBlock:^id(AWSTask *task) {
    if (task.error) {
        NSLog(@"The request failed. Error: [%@]", task.error);
    } else {
        AWSDynamoDBPaginatedOutput *paginatedOutput = task.
↪result;
        for (Book *book in paginatedOutput.items) {
            //Do something with book.
        }
    }
```

```
    return nil;
}];
```

Filter a Scan

The output of a scan is returned as an `AWSDynamoDBPaginatedOutput` object. The array of returned items is in the `items` property.

The `scanExpression` method provides several optional parameters. Use `filterExpression` and `expressionAttributeValues` to specify a scan result for the attribute names and conditions you define. For more information about the parameters and the API, see AWSDynamoDBScanExpression.

The following code scans the Books table to find books with a price less than 50.

Swift

```
let scanExpression = AWSDynamoDBScanExpression()
scanExpression.limit = 10
scanExpression.filterExpression = "Price < :val"
scanExpression.expressionAttributeValues = [":val": 50]

dynamoDBObjectMapper.scan(Book.self, expression: scanExpression).
↪continueWith(block: { (task:AWSTask<AnyObject>!) -> Any? in
  if let error = task.error as? NSError {
      print("The request failed. Error: \(error)")
  } else if let paginatedOutput = task.result {
      for book in paginatedOutput.items as! Book {
          // Do something with book.
      }
  }
})
```

Objective-C

```
AWSDynamoDBScanExpression *scanExpression =␣
↪[AWSDynamoDBScanExpression new];
scanExpression.limit = @10;
scanExpression.filterExpression = @"Price < :val";
scanExpression.expressionAttributeValues = @{@":val":@50};

[[dynamoDBObjectMapper scan:[Book class]
            expression:scanExpression]
continueWithBlock:^id(AWSTask *task) {
    if (task.error) {
        NSLog(@"The request failed. Error: [%@]", task.error);
    } else {
        AWSDynamoDBPaginatedOutput *paginatedOutput = task.
↪result;
        for (Book *book in paginatedOutput.items) {
            //Do something with book.
        }
```

```
    }
    return nil;
}];
```

You can also use the `projectionExpression` property to specify the attributes to retrieve from the ``Books` table. For example adding `scanExpression.projectionExpression = @"ISBN,Title,Price";` in the previous code snippet retrieves only those three properties in the book object. The `Author` property in the book object will always be nil.

4.2.5 Perform a Query

The query API enables you to query a table or a secondary index. The `query:expression:` method takes two parameters: the class of the resulting object and an instance of `AWSDynamoDBQueryExpression`.

To query an index, you must also specify the `indexName`. You must specify the `hashKeyAttribute` if you query a global secondary with a different `hashKey`. If the table or index has a range key, you can optionally refine the results by providing a range key value and a condition.

The following example illustrates querying the *Books* index table to find all books whose author is "John Smith", with a price less than 50.

Swift

```
let queryExpression = AWSDynamoDBQueryExpression()
queryExpression.indexName = "Author-Price-index"

queryExpression.keyConditionExpression = @"Author = :authorName␣
↪AND Price < :val";
queryExpression.expressionAttributeValues = @{@":authorName": @
↪"John Smith", @":val": @50};

dynamoDBObjectMapper.query(Book.self, expression:␣
↪queryExpression).continueWith(block: { (task:AWSTask<AnyObject>
↪!) -> Any? in
    if let error = task.error as? NSError {
          print("The request failed. Error: \(error)")
    } else if let paginatedOutput = task.result {
        for book in paginateOutput.items as! Book {
            // Do something with book.
        }
    }
    return nil
})
```

Objective-C

```
AWSDynamoDBQueryExpression *queryExpression =␣
↪[AWSDynamoDBQueryExpression new];
```

```
queryExpression.indexName = @"Author-Price-index";

queryExpression.keyConditionExpression = @"Author = :authorName␣
↪AND Price < :val";

queryExpression.expressionAttributeValues = @{@":authorName": @
↪"John Smith", @":val":@50};

[[dynamoDBObjectMapper query:[Book class]
        expression:queryExpression]
continueWithBlock:^id(AWSTask *task) {
     if (task.error) {
         NSLog(@"The request failed. Error: [%@]", task.error);
     } else {
         AWSDynamoDBPaginatedOutput *paginatedOutput = task.
↪result;
         for (Book *book in paginatedOutput.items) {
             //Do something with book.
         }
     }
     return nil;
 }];
```

In the preceding example, `indexName` is specified to demonstrate querying an index. The query expression is specified using `keyConditionExpression` and the values used in the expression using `expressionAttributeValues`.

You can also provide `filterExpression` and ‘‘projectionExpressio‘n‘ in ‘‘AWSDynamoDBQueryExpression‘. The syntax is the same as that used in a scan operation.

For more information, see AWSDynamoDBQueryExpression. *Migrating AWSDynamoDBModel to AWSDynamoDBObjectModel*

As of SDK version 2.0.16, the `AWSDynamoDBModel` mapping class is deprecated and replaced by `AWSDynamoDBObjectModel`.The deprecated `AWSDynamoDBModel` used *NSArray* to represent multi-valued types (`String Set`, `Number Set`, and `Binary Set`); it did not support `Boolean`, `Map`, or `List` types. The new `AWSDynamoDBObjectModel` uses `NSSet` for multi-valued types and supports `Boolean`, `Map`, and `List`. For the `Boolean` type, you create an `NSNumber` using `[NSNumber numberWithBool:YES]` or using the shortcuts `@YES` and `@NO`. For the Map type, create using `NSDictionary`. For the List type, create using `NSArray`.

4.2.6 Additional Resources

- Amazon DynamoDB Developer Guide
- Amazon DynamoDB API Reference

4.3 Amazon DynamoDB Low-level Client

On this page:

Amazon DynamoDB is a fast, highly scalable, highly available, cost-effective, nonrelational database service. Amazon DynamoDB removes traditional scalability limitations on data storage while maintaining low latency and predictable performance.

The AWS Mobile SDK for iOS provides both low-level and high-level libraries for working Amazon DynamoDB.

The low-level client described in this section allows the kind of direct access to Amazon DynamoDB tables useful for NoSQL and other non-relational data designs. The low-level client also supports conditional data writes to mitigate simultaneous write conflicts and batch data writes.

The high-level library includes *Amazon DynamoDB Object Mapper API*, which lets you map client-side classes to access and manipulate Amazon Dynamo tables.

4.3.1 Setup

To set your project up to use the AWS SDK for iOS TransferUtility, take the following steps.

1. Setup the SDK, Credentials, and Services

Follow the steps described in *Amazon DynamoDB Integration Setup* to install the AWS Mobile SDK for iOS and configure AWS services, credentials, and permissions.

2. Import the AWSDynamoDB APIs

Add the following import statement to your project.

Swift

```
import AWSDynamoDB
```

Objective-C

```
#import <AWSDynamoDB/AWSDynamoDB.h>
```

4.3.2 Conditional Writes Using the Low-Level Client

In a multi-user environment, multiple clients can access the same item and attempt to modify its attribute values at the same time. To help clients coordinate writes to data items, the Amazon DynamoDB low-level

client supports conditional writes for *PutItem*, *DeleteItem*, and *UpdateItem* operations. With a conditional write, an operation succeeds only if the item attributes meet one or more expected conditions; otherwise, it returns an error.

In the following example, we update the price of an item in the `Books` table if the `Price` attribute of the item has a value of `999`.

Swift

```
Amazon DynamoDB = AWSDynamoDB.default()
let updateInput = AWSDynamoDBUpdateItemInput()

let hashKeyValue = AWSDynamoDBAttributeValue()
hashKeyValue?.s = "4567890123"

updateInput?.tableName = "Books"
updateInput?.key = ["ISBN": hashKeyValue!]

let oldPrice = AWSDynamoDBAttributeValue()
oldPrice?.n = "999"

let expectedValue = AWSDynamoDBExpectedAttributeValue()
expectedValue?.value = oldPrice

let newPrice = AWSDynamoDBAttributeValue()
newPrice?.n = "1199"

let valueUpdate = AWSDynamoDBAttributeValueUpdate()
valueUpdate?.value = newPrice
valueUpdate?.action = .put

updateInput?.attributeUpdates = ["Price": valueUpdate!]
updateInput?.expected = ["Price": expectedValue!]
updateInput?.returnValues = .updatedNew

Amazon DynamoDB.updateItem(updateInput!).continueWith { (task:
↪AWSTask<AWSDynamoDBUpdateItemOutput>) -> Any? in
    if let error = task.error as? NSError {
        print("The request failed. Error: \(error)")
        return nil
    }

    // Do something with task.result

    return nil
}
```

Objective-C

```
AWSDynamoDB *dynamoDB = [AWSDynamoDB defaultDynamoDB];
AWSDynamoDBUpdateItemInput *updateInput =␣
↪[AWSDynamoDBUpdateItemInput new];

AWSDynamoDBAttributeValue *hashKeyValue =␣
↪[AWSDynamoDBAttributeValue new];
```

```
hashKeyValue.S = @"4567890123";

updateInput.tableName = @"Books";
updateInput.key = @{ @"ISBN" : hashKeyValue };

AWSDynamoDBAttributeValue *oldPrice = [AWSDynamoDBAttributeValue␣
↪new];
oldPrice.N = @"999";

AWSDynamoDBExpectedAttributeValue *expectedValue =␣
↪[AWSDynamoDBExpectedAttributeValue new];
expectedValue.value = oldPrice;

AWSDynamoDBAttributeValue *newPrice = [AWSDynamoDBAttributeValue␣
↪new];
newPrice.N = @"1199";

AWSDynamoDBAttributeValueUpdate *valueUpdate =␣
↪[AWSDynamoDBAttributeValueUpdate new];
valueUpdate.value = newPrice;
valueUpdate.action = AWSDynamoDBAttributeActionPut;

updateInput.attributeUpdates = @{@"Price": valueUpdate};
updateInput.expected = @{@"Price": expectedValue};
updateInput.returnValues = AWSDynamoDBReturnValueUpdatedNew;

[[dynamoDB updateItem:updateInput]
 continueWithBlock:^id(AWSTask *task) {
     if (task.error) {
         NSLog(@"The request failed. Error: [%@]", task.error);
     } else {
         //Do something with task.result.
     }
     return nil;
 }];
```

Conditional writes are idempotent. In other words, if a conditional write request is made multiple times, the update will be performed only in the first instance unless the content of the request changes. In the preceding example, sending the same request a second time results in a *ConditionalCheckFailedException*, because the expected condition is not met after the first update.

4.3.3 Batch Operations Using the Low-Level Client

The Amazon DynamoDB low-level client provides batch write operations to put items in the database and delete items from the database. You can also use batch get operations to return the attributes of one or more items from one or more tables.

The following example shows a batch write operation.

Swift

```
Amazon DynamoDB = AWSDynamoDB.default()

//Write Request 1
let hashValue1 = AWSDynamoDBAttributeValue()
hashValue1?.s = "3210987654"
let otherValue1 = AWSDynamoDBAttributeValue()
otherValue1?.s = "Some Title"

let writeRequest = AWSDynamoDBWriteRequest()
writeRequest?.putRequest = AWSDynamoDBPutRequest()
writeRequest?.putRequest?.item = ["ISBN": hashValue1!, "Title":␣
↪otherValue1!]

//Write Request 2
let hashValue2 = AWSDynamoDBAttributeValue()
hashValue2?.s = "8901234567"
let otherValue2 = AWSDynamoDBAttributeValue()
otherValue2?.s = "Another Title"

let writeRequest2 = AWSDynamoDBWriteRequest()
writeRequest2?.putRequest = AWSDynamoDBPutRequest()
writeRequest2?.putRequest?.item = ["ISBN": hashValue2!, "Title":
↪ otherValue2!]

let batchWriteItemInput = AWSDynamoDBBatchWriteItemInput()
batchWriteItemInput?.requestItems = ["Books": [writeRequest!,␣
↪writeRequest2!]]

Amazon DynamoDB.batchWriteItem(batchWriteItemInput!).
↪continueWith { (task:AWSTask<AWSDynamoDBBatchWriteItemOutput>
↪) -> Any? in
    if let error = task.error as? NSError {
        print("The request failed. Error: \(error)")
        return nil
    }

    // Do something with task.result

    return nil
}
```

Objective-C

```
AWSDynamoDB *dynamoDB = [AWSDynamoDB defaultDynamoDB];

//Write Request 1
AWSDynamoDBAttributeValue *hashValue1 =␣
↪[AWSDynamoDBAttributeValue new];
hashValue1.S = @"3210987654";
AWSDynamoDBAttributeValue *otherValue1 =␣
↪[AWSDynamoDBAttributeValue new];
otherValue1.S = @"Some Title";
```

```
AWSDynamoDBWriteRequest *writeRequest = [AWSDynamoDBWriteRequest␣
↪new];
writeRequest.putRequest = [AWSDynamoDBPutRequest new];
writeRequest.putRequest.item = @{
                                  @"ISBN" : hashValue1,
                                  @"Title" : otherValue1
                                  };

//Write Request 2
AWSDynamoDBAttributeValue *hashValue2 =␣
↪[AWSDynamoDBAttributeValue new];
hashValue2.S = @"8901234567";
AWSDynamoDBAttributeValue *otherValue2 =␣
↪[AWSDynamoDBAttributeValue new];
otherValue2.S = @"Another Title";

AWSDynamoDBWriteRequest *writeRequest2 =␣
↪[AWSDynamoDBWriteRequest new];
writeRequest2.putRequest = [AWSDynamoDBPutRequest new];
writeRequest2.putRequest.item = @{
                                  @"ISBN" : hashValue2,
                                  @"Title" : otherValue2
                                  };

AWSDynamoDBBatchWriteItemInput *batchWriteItemInput =␣
↪[AWSDynamoDBBatchWriteItemInput new];
batchWriteItemInput.requestItems = @{@"Books":␣
↪@[writeRequest,writeRequest2]};

[[dynamoDB batchWriteItem:batchWriteItemInput]
 continueWithBlock:^id(AWSTask *task) {
     if (task.error) {
         NSLog(@"The request failed. Error: [%@]", task.error);
     } else  {
        //Do something with task.result.
    }
    return nil;
}];
```

4.3.4 Additional Resources

- Amazon DynamoDB Developer Guide
- Amazon DynamoDB API Reference

Amazon DynamoDB is a fast, highly scalable, highly available, cost-effective, non-relational database service. Amazon DynamoDB removes traditional scalability limitations on data storage while maintaining low latency and predictable performance.

The AWS Mobile SDK for iOS provides both low-level and high-level libraries for working with Amazon DynamoDB. Both allow you to perform create, read, update, and delete (CRUD) operations and to execute queries and scans.

The high-level library includes Amazon DynamoDB object mapper which lets you map client-side classes to tables. Working within the data model defined on your client you can write simple, readable code that stores and retrieves objects in the cloud. See *Amazon DynamoDB Object Mapper API*.

The low-level client allows you to access and manipulate Amazon DynamoDB tables directly for NoSQL or other non-relational data designs. The low-level library also supports conditional writes, to mitigate issues with simultaneous writes by multiple users, and batch operations. See *Amazon DynamoDB Low-level Client*.

4.4 Get Started

You can explore Amazon DynamoDB in the following ways:

- *Amazon DynamoDB Integration Setup*: Use the walkthroughs provided in this SDK.
- Try the NoSQL Database feature in the AWS Mobile Hub console:
  - Intuitively design and provisioning of AWS services in minutes
  - Quickstart app demonstrating the services you configure (Android, Swift, or Objective-C)
  - SDK with helper code that can be dropped into your Xcode project
  - Step-by-step instructions for integrating your services into an existing mobile app

  Sign in with your AWS account and create a Mobile hub project - both free - then select NoSQL Database.
- DynamoDB sample: Download and build the sample app from Github

4.5 Additional Resources

- For information Amazon DynamoDB Region availability, see AWS Service Region Availability.
- Amazon DynamoDB Developer Guide
- Amazon DynamoDB API Reference

CHAPTER 5

AWS Lambda: Execute Code On Demand

On this page:

Amazon Lambda

The AWS Lambda service makes it easy to create scalable, secure, and highly available backends for your mobile apps without the need to provision or manage infrastructure.

You can create secure logical functions in the cloud that can be called directly from your iOS app. Your AWS Lambda code, written in C#, Node.js, Python, or Java, can implement standalone logic, extend your app to a range of AWS services, and/or connect to services and applications external to AWS.

The availability and cost of a AWS Lambda function automatically scales to amount of traffic it receives. Functions can also be accessed from an iOS app through Amazon API Gateway, giving features like global provisioning, enterprise grade monitoring, throttling and control of access.

5.1 Setup

This section provides a step-by-step guide for getting started with AWS Lambda using the AWS Mobile SDK for iOS.

1. Install the SDK

   Add the AWS SDK for iOS to your project and import the APIs you need, by following the steps described in *Set Up the SDK for iOS*.

2. Configure Credentials

   To use Amazon Cognito to create AWS identities and credentials that give your users access to your app's AWS resources, follow the steps described at *Amazon Cognito for iOS*.

3. Create and Configure a Lamda Function

   (a) Sign in to the AWS Lambda console.

   (b) Choose *Create a Lamda function*.

   (c) Choose the *Blank Function* template.

   Note that dozens of function templates that connect to other AWS services are available.

   (d) Choose *Next*.

   Note that the console allows you to configure triggers for a function from other AWS services, these won't be used in this walkthrough.

   (e) Type a *Name* and select *Node.js* as the *Runtime* language.

   (f) Under *Lambda function handler and role*, select *Create new role from template(s)*. Type a *Role name*. Select the *Policy template* named *Simple Microservice permissions*.

   (g) Choose *Next*.

   (h) Choose *Create function*.

5.2 Invoking an AWS Lambda Function

The SDK enables you to call AWS Lambda functions from your iOS mobile apps, using the AWSLambdaInvoker class. When invoked from this SDK, AWS Lambda functions receive data about the device and the end user identity through client and identity context objects. To learn more about using these contexts to create rich, and personalized app experiences, see *Client Context* and *Identity Context*.

5.2.1 Import AWS Lambda API

To use the *lambdainvoker* API, use the following import statement:

Swift

```
import AWSLambda
```

Objective C

```
#import <AWSLambda/AWSLambda.h>
```

5.2.2 Call lambdaInvoker

`AWSLambdaInvoker` provides a high-level abstraction for AWS Lambda. When `invokeFunction JSONObject` is invoked, the JSON object is serialized into JSON data and sent to the AWS Lambda service. AWS Lambda returns a JSON encoded response that is deserialized into a JSON object.

A valid JSON object must have the following properties:

- All objects are instances of string, number, array, dictionary or null objects.

- All dictionary keys are instances of string objects.
- Numbers are not `NaN` or `infinity`.

The following is an example of valid request.

Swift

```
let lambdaInvoker = AWSLambdaInvoker.default()
let jsonObject: [String: Any] = ["key1" : "value1",
                         "key2" : 2 ,
                         "key3" : [1, 2],
                         "isError" : false]

lambdaInvoker.invokeFunction("myFunction", jsonObject:␣
↪jsonObject)
    .continueWith(block: {(task):AWSTask<AnyObject>) -> Any? in
    if( let error = task.error != nil) {as? NSError {
        print(task.("Error: \(error!)")")
        return nil
    }

    // Handle response in task.result
    return nil
})
```

Objective C

```
AWSLambdaInvoker *lambdaInvoker = [AWSLambdaInvoker␣
↪defaultLambdaInvoker];

[[lambdaInvoker invokeFunction:@"myFunction"
            JSONObject:@{@"key1" : @"value1",
                         @"key2" : @2,
                         @"key3" : [NSNull null],
                         @"key4" : @[@1, @"2"],
                         @"isError" : @NO}] continueWithBlock:^
↪id(AWSTask *task) {
    // Handle response
    return nil;
}];
```

5.2.3 Using function returns

On successful execution, *task.result* contains a JSON object. For instance, if *myFunctions* returns a dictionary, you can cast the result to a dictionary object as follows.

Swift

```
if let JSONDictionary = task.result as? NSDictionary {
    print("Result: \(JSONDictionary)")
    print("resultKey: \(JSONDictionary["resultKey"])")
}
```

Objective C

```
if (task.result) {
    NSLog(@"Result: %@", task.result);
    NSDictionary *JSONObject = task.result;
    NSLog(@"result: %@", JSONObject[@"resultKey"]);
}
```

5.2.4 Handling service execution errors

On failed AWS Lambda service execution, *task.error* may contain a *NSError* with *AWSLambdaErrorDomain* domain and the following error code.

- *AWSLambdaErrorUnknown*
- *AWSLambdaErrorService*
- *AWSLambdaErrorResourceNotFound*
- *AWSLambdaErrorInvalidParameterValue*

On failed function execution, *task.error* may contain a *NSError* with *AWSLambdaInvokerErrorDomain* domain and the following error code:

- *AWSLambdaInvokerErrorTypeUnknown*
- *AWSLambdaInvokerErrorTypeFunctionError*

When *AWSLambdaInvokerErrorTypeFunctionError* error code is returned, *error.userInfo* may contain a function error from your AWS Lambda function with *AWSLambdaInvokerFunctionErrorKey* key.

The following code shows error handling.

Swift

```
if let error = task.error as? NSError {
    if error.domain == AWSLambdaInvokerErrorDomain &&␣
↪AWSLambdaInvokerErrorType.functionError ==␣
↪AWSLambdaInvokerErrorType(rawValue: error.code) {
        print("Function error: \(error.
↪userInfo[AWSLambdaInvokerFunctionErrorKey])")
    } else {
        print("Error: \(error)")
    }
    return nil
}
```

Objective C

```
if (task.error) {
    NSLog(@"Error: %@", task.error);
    if ([task.error.domain isEqualToString:
↪AWSLambdaInvokerErrorDomain]
        && task.error.code ==␣
↪AWSLambdaInvokerErrorTypeFunctionError) {
```

```
        NSLog(@"Function error: %@", task.error.
↪userInfo[AWSLambdaInvokerFunctionErrorKey]);
    }
}
```

5.2.5 Comprehensive example

The following code shows invoking an AWS Lambda call and handling returns and errors all together.

Swift

```
let lambdaInvoker = AWSLambdaInvoker.default()

let jsonObject: [String: Any] = ["key1" : "value1",
                        "key2" : 2,
                        "key3" : [1, 2],
                        "isError" : false]

lambdaInvoker.invokeFunction("myFunction", jsonObject:␣
↪jsonObject).continueWith(block: {(task:AWSTask<AnyObject>) ->␣
↪Any? in
    if let error = task.error as? NSError {
        if error.domain == AWSLambdaInvokerErrorDomain &&␣
↪AWSLambdaInvokerErrorType.functionError ==␣
↪AWSLambdaInvokerErrorType(rawValue: error.code) {
            print("Function error: \(error.
↪userInfo[AWSLambdaInvokerFunctionErrorKey])")
        } else {
            print("Error: \(error)")
        }
        return nil
    }

    // Handle response in task.result
    if let JSONDictionary = task.result as? NSDictionary {
        print("Result: \(JSONDictionary)")
        print("resultKey: \(JSONDictionary["resultKey"])")
    }
    return nil
})
```

Objective C

```
AWSLambdaInvoker *lambdaInvoker = [AWSLambdaInvoker␣
↪defaultLambdaInvoker];

[[lambdaInvoker invokeFunction:@"myFunction"
            JSONObject:@{@"key1" : @"value1",
                         @"key2" : @2,
                         @"key3" : [NSNull null],
                         @"key4" : @[@1, @"2"],
                         @"isError" : @NO}] continueWithBlock:^
↪id(AWSTask *task) {
```

```
    if (task.error) {
        NSLog(@"Error: %@", task.error);
        if ([task.error.domain isEqualToString:
↪AWSLambdaInvokerErrorDomain]
            && task.error.code ==␣
↪AWSLambdaInvokerErrorTypeFunctionError) {
            NSLog(@"Function error: %@", task.error.
↪userInfo[AWSLambdaInvokerFunctionErrorKey]);
        }
    }
    if (task.result) {
        NSLog(@"Result: %@", task.result);
        NSDictionary *JSONObject = task.result;
        NSLog(@"result: %@", JSONObject[@"resultKey"]);
    }
    return nil;
}];
```

5.3 Client Context

Calls to AWS Lambda using this SDK provide your functions with data about the calling device and app using the *ClientContext* class.

You can access the client context in your lambda function as follows.

JavaScript

```
exports.handler = function(event, context) {
    console.log("installation_id = " + context.clientContext.
↪client.installation_id);
    console.log("app_version_code = " + context.clientContext.
↪client.app_version_code);
    console.log("app_version_name = " + context.clientContext.
↪client.app_version_name);
    console.log("app_package_name = " + context.clientContext.
↪client.app_package_name);
    console.log("app_title = " + context.clientContext.client.
↪app_title);
    console.log("platform_version = " + context.clientContext.
↪env.platform_version);
    console.log("platform = " + context.clientContext.env.
↪platform);
    console.log("make = " + context.clientContext.env.make);
    console.log("model = " + context.clientContext.env.model);
    console.log("locale = " + context.clientContext.env.locale);

    context.succeed("Your platform is " + context.clientContext.
↪env.platform;
}
```

ClientContext has the following fields:

client.installation_id Auto-generated UUID that is created the first time the app is launched. This is stored in the keychain on the device. In case the keychain is wiped a new installation ID will be generated.

client.app_version_code CFBundleShortVersionString

client.app_version_name CFBundleVersion

client.app_package_name CFBundleIdentifier

client.app_title CFBundleDisplayName

env.platform_version systemVersion

env.platform systemName

env.make Hardcoded as "apple"

env.model Model of the device

env.locale localeIdentifier from autoupdatingCurrentLocale

5.4 Identity Context

The *IdentityContext* class of the SDK passes Amazon Cognito credentials making the AWS identity of the end user available to your function. You can access the Identity ID as follows.

JavaScript

```
exports.handler = function(event, context) {
    console.log("clientID = " + context.identity);

    context.succeed("Your client ID is " + context.identity);
}
```

For more about Amazon Cognito in the AWS Mobile SDK for iOS, see *Amazon Cognito for iOS*.

CHAPTER 6

Amazon Cognito Sync: Sync User Data

6.1 Authenticate Users with Amazon Cognito Identity

Amazon Cognito Identity provides secure access to AWS services. Identities are managed by an identity pool. Roles specify resources an identity can access and are associated with an identity pool. To create an identity pool for your application:

1. Log into the **'Amazon Cognito Console'_** and click the *New Identity Pool* button
2. Give your Identity Pool a unique name and enable access to unauthenticated identities
3. Click the *Create Pool* button and then the *Update Roles* to create your identity pool and associated roles

For more information on Amazon Cognito Identity, see *Amazon Cognito for iOS*

Note: The auto-generated Roles include the permissions needed to access Amazon Cognito Sync, so no further configuration is required.

The next page displays code that creates a credential provider that provides a Amazon Cognito Identity for your app to use. Copy the code from Steps 1 & 2 into your AppDelegate.m file as shown below:

Add the following import statements:

Swift

```
import AWSCore
import AWSCognito
```

Objective-C

```
#import <AWSCore/AWSCore.h>
#import <AWSCognito/AWSCognito.h>
```

If you have an existing AWS credential provider, add the following code to *application:didFinishLaunchingWithOptions* method:

Swift

```
let credentialProvider =␣
↪AWSCognitoCredentialsProvider(regionType: .USEast1,␣
↪identityPoolId: "YourIdentityPoolId")
let configuration = AWSServiceConfiguration(region: .USEast1,␣
↪credentialsProvider: credentialProvider)
AWSServiceManager.default().defaultServiceConfiguration =␣
↪configuration
```

Objective-C

```
AWSCognitoCredentialsProvider *credentialsProvider =␣
↪[[AWSCognitoCredentialsProvider alloc] initWithRegionType:
↪AWSRegionUSEast1
identityPoolId:@"<your-identity-pool-arn>"];

AWSServiceConfiguration *configuration =␣
↪[[AWSServiceConfiguration alloc] initWithRegion:
↪AWSRegionUSEast1 credentialsProvider:credentialsProvider];

AWSServiceManager.defaultServiceManager.
↪defaultServiceConfiguration = configuration;
```

For more information on Amazon Cognito Identity, see *Amazon Cognito for iOS*

6.2 Syncing User Data

To sync unauthenticated user data:

1. Create a dataset and add user data.
2. Synchronize the dataset with the cloud.

6.3 Create a Dataset and Add User Data

Create an instance of `AWSCognitoDataset`. User data is added in the form of key/value pairs. Dataset objects are created with the `AWSCognito` class which functions as a Amazon Cognito client object. Use the defaultCognito method to get a reference to the default singleton instance of AWSCognito. The openOrCreateDataset method is used to create a new dataset or open an existing instance of a dataset stored locally on the device:

Swift

```
let dataset = AWSCognito.default().openOrCreateDataset("user_data
↪")
```

Objective-C

```
AWSCognitoDataset *dataset = [[AWSCognito defaultCognito]␣
↪openOrCreateDataset:datasetName];:@"user_data"];
```

User data is added to an AWSCognitoDataset instance using the setString:forKey or setValue:forKey methods. The following code snippet shows how to add some user data to a dataset:

Swift

```
dataset?.setString("John Doe", forKey:"Username")
dataset?.setString("10000", forKey:"HighScore")
```

Objective-C

```
[dataset setString:@"John Doe" forKey:@"Username"];
[dataset setString:@"10000" forKey:@"HighScore"];
```

6.4 Synchronize Dataset with the Cloud

To sync the dataset with the cloud, call the synchronize method on the dataset object:

Swift

```
_ = dataset?.synchronize()
```

Objective-C

```
[dataset synchronize];
```

All data written to datasets will be stored locally until the dataset is synced. The code in this section assumes you are using an unauthenticated Amazon Cognito identity, so when the user data is synced with the cloud it will be stored per device. The device has a device ID associated with it, when the user data is synced to the cloud, it will be associated with that device ID.

To sync user data across devices (based on an authenticated Cognito Identity) see Amazon Cognito Sync Developer Guide.

6.5 Related Documentation

Amazon Cognito for iOS

Developer Authenticated Identities

CHAPTER 7

Amazon Kinesis and Amazon Kinesis Firehose: Process Streaming Data

Amazon Kinesis is a fully managed service for real-time processing of streaming data at massive scale.

The SDK for iOS provides two high-level client classes, **`AWSKinesisRecorder`** and **`AWSFirehoseRecorder`**, designed to help you interface with Amazon Kinesis and Amazon Kinesis Firehose.

The Amazon Kinesis **`AWSKinesisRecorder`** client lets you store PutRecord requests on disk and then send them all at once. This is useful because many mobile applications that use Amazon Kinesis will create multiple **`PutRecord`** requests per second. Sending an individual request for each **`PutRecord`** action could adversely impact battery life. Moreover, the requests could be lost if the device goes offline. Thus, using the high-level Amazon Kinesis client for batching can preserve both battery life and data.

The Amazon Kinesis Firehose **`AWSFirehoseRecorder`** client lets you store PutRecords requests on disk and then send them using **`PutRecordBatch`**.

For information about Amazon Kinesis Region availability, see AWS Service Region Availability.

7.1 What is Amazon Kinesis?

Amazon Kinesis is a fully managed service for real-time processing of streaming data at massive scale. Amazon Kinesis can collect and process hundreds of terabytes of data per hour from hundreds of thousands of sources, so you can write applications that process information in real-time. With Amazon Kinesis applications, you can build real-time dashboards, capture exceptions and generate alerts, drive recommendations, and make other real-time business or operational decisions. You can also easily send data to other services such as Amazon Simple Storage Service, Amazon DynamoDB, and Amazon Redshift.

7.2 What is Amazon Kinesis Firehose?

Amazon Kinesis Firehose is a fully managed service for delivering real-time streaming data to destinations such as Amazon Simple Storage Service (Amazon S3) and Amazon Redshift. With Firehose, you do not

need to write any applications or manage any resources. You configure your data producers to send data to Firehose and it automatically delivers the data to the destination that you specified.

For more information about Amazon Kinesis Firehose, see Amazon Kinesis Firehose.

You can also learn more about how the Amazon Kinesis services work together on the following page: Amazon Kinesis services.

7.3 Integrating Amazon Kinesis and Amazon Kinesis Firehose

To use the Amazon Kinesis mobile client, you'll need to integrate the SDK for iOS into your app and import the necessary libraries. To do so, follow these steps:

If you haven't already done so, download the SDK for iOS, unzip it, and include it in your application as described at *Set Up the SDK for iOS*. The instructions direct you to import the headers for the services you'll be using. For this example, you need the following import.

Swift

```
import AWSKinesis
```

Objective-C

```
#import <AWSKinesis/AWSKinesis.h>
```

You can use Amazon Cognito to provide temporary AWS credentials to your application.

These credentials let the app access your AWS resources. To create a credentials provider, follow the instructions at Cognito Identity Developer Guide.

To use Amazon Kinesis in an application, you must set the correct permissions. The following IAM policy allows the user to submit records to a specific Amazon Kinesis stream, which is identified by ARN.

```
{
    "Statement": [{
        "Effect": "Allow",
        "Action": "kinesis:PutRecords",
        "Resource": "arn:aws:kinesis:us-west-2:111122223333:stream/
↪mystream"
    }]
}
```

The following IAM policy allows the user to submit records to a specific Amazon Kinesis Firehose stream.

```
{
    "Statement": [{
        "Effect": "Allow",
        "Action": "firehose:PutRecordBatch",
        "Resource": "arn:aws:firehose:us-west-2:111122223333:
↪deliverystream/mystream"
    }]
}
```

This policy should be applied to roles assigned to the Amazon Cognito identity pool, but you will need to replace the **Resource** value with the correct ARN for your Amazon Kinesis or Amazon Kinesis Firehose stream. You can apply policies at the IAM console. To learn more about IAM policies, see Using IAM.

To learn more about Amazon Kinesis-specific policies, see Controlling Access to Amazon Kinesis Resources with IAM.

To learn more about Amazon Kinesis Firehose policies, see Controlling Access with Amazon Kinesis Firehose.

Once you have credentials, you can use **AWSKinesisRecorder** with Amazon Kinesis. The following snippet returns a shared instance of the Amazon Kinesis service client:

Swift

```
let kinesisRecorder = AWSKinesisRecorder.default()
```

Objective-C

```
AWSKinesisRecorder *kinesisRecorder = [AWSKinesisRecorder␣
↪defaultKinesisRecorder];
```

You can use **AWSFirehoseRecorder** with Amazon Kinesis Firehose. The following snippet returns a shared instance of the Amazon Kinesis Firehose service client:

Swift

```
let firehoseRecorder = AWSFirehoseRecorder.default()
```

Objective-C

```
AWSFirehoseRecorder *firehoseRecorder = [AWSFirehoseRecorder␣
↪defaultFirehoseRecorder];
```

You can configure **AWSKinesisRecorder** or **AWSFirehoseRecorder** through their properties:

Swift

```
kinesisRecorder.diskAgeLimit = TimeInterval(30 * 24 * 60 * 60); /
↪/ 30 days
kinesisRecorder.diskByteLimit = UInt(10 * 1024 * 1024); // 10MB
kinesisRecorder.notificationByteThreshold = UInt(5 * 1024 *␣
↪1024); // 5MB
```

Objective-C

```
kinesisRecorder.diskAgeLimit = 30 * 24 * 60 * 60; // 30 days
kinesisRecorder.diskByteLimit = 10 * 1024 * 1024; // 10MB
kinesisRecorder.notificationByteThreshold = 5 * 1024 * 1024; //␣
↪5MB
```

The **diskAgeLimit** property sets the expiration for cached requests. When a request exceeds the limit, it's discarded. The default is no age limit. The **diskByteLimit** property holds the limit of the disk cache size in bytes. If the storage limit is exceeded, older requests are discarded. The default value is 5

MB. Setting the value to 0 means that there's no practical limit. The **notficationByteThreshold** property sets the point beyond which Kinesis issues a notification that the byte threshold has been reached. The default value is 0, meaning that by default Amazon Kinesis doesn't post the notification.

To see how much local storage is being used for Amazon Kinesis **PutRecord** requests, check the **diskBytesUsed** property.

With **AWSKinesisRecorder** created and configured, you can use **saveRecord:streamName:** to save records to local storage.

In the preceding example, we create an NSData object and save it locally. **YourStreamName** should be a string corresponding to the name of your Kinesis stream. You can create new streams in the Amazon Kinesis console.

Here is a similar snippet for Amazon Kinesis Firehose:

Swift

```
let yourData = "Test_data".data(using: .utf8)
firehoseRecorder.saveRecord(yourData, streamName: "YourStreamName
↪")
```

Objective-C

```
NSData *yourData = [@"Test_data" dataUsingEncoding:
↪NSUTF8StringEncoding];
[firehoseRecorder saveRecord:yourData streamName:@"YourStreamName
↪"]
```

To submit all the records stored on the device, call **submitAllRecords**.

Swift

```
kinesisRecorder.submitAllRecords()

firehoseRecorder.submitAllRecords()
```

Objective-C

```
[kinesisRecorder submitAllRecords];

[firehoseRecorder submitAllRecords];
```

submitAllRecords sends all locally saved requests to the Amazon Kinesis service. Requests that are successfully sent will be deleted from the device. Requests that fail because the device is offline will be kept and submitted later. Invalid requests are deleted.

Both **saveRecord** and **submitAllRecords** are asynchronous operations, so you should ensure that **saveRecord** is complete before you invoke **submitAllRecords**. The following code sample shows the methods used correctly together.

Swift

```
// Create an array to store a batch of objects.
var tasks = Array<AWSTask<AnyObject>>()
for i in 0...100 {
    tasks.append(kinesisRecorder!.saveRecord(String(format:
↪"TestString-%02d", i).data(using: .utf8), streamName:
↪"YourStreamName")!)
}

AWSTask(forCompletionOfAllTasks: tasks).
↪continueOnSuccessWith(block: { (task:AWSTask<AnyObject>) ->␣
↪AWSTask<AnyObject>? in
    return kinesisRecorder?.submitAllRecords()
}).continueWith(block: { (task:AWSTask<AnyObject>) -> Any? in
    if let error = task.error as? NSError {
        print("Error: \(error)")
    }
    return nil
})
```

Objective-C

```
// Create an array to store a batch of objects.
NSMutableArray *tasks = [NSMutableArray new];
for (int32_t i = 0; i < 100; i++) {
    [tasks addObject:[kinesisRecorder saveRecord:[[NSString␣
↪stringWithFormat:@"TestString-%02d", i] dataUsingEncoding:
↪NSUTF8StringEncoding]
                                    streamName:@"YourStreamName"]];
}
[[[AWSTask taskForCompletionOfAllTasks:tasks]␣
↪continueWithSuccessBlock:^id(AWSTask *task) {
    return [kinesisRecorder submitAllRecords];
}] continueWithBlock:^id(AWSTask *task) {
    if (task.error) {
        NSLog(@"Error: [%@]", task.error);
    }
    return nil;
}];
```

To learn more about working with Amazon Kinesis, see the Amazon Kinesis Developer Resources.

To learn more about the Amazon Kinesis classes, see the class reference for AWSKinesisRecorder.

For information about AWS service region availability, see AWS Service Region Availability.

CHAPTER 8

Amazon Lex: Use Natural Language to Trigger Business Workflows

8.1 What is Amazon Lex?

Amazon Lex is an AWS service for building voice and text conversational interfaces into applications. With Amazon Lex, the same natural language understanding engine that powers Amazon Alexa is now available to any developer, enabling you to build sophisticated, natural language chatbots into your new and existing applications.

The AWS Mobile SDK for iOS provides an optimized client for interacting with Amazon Lex runtime APIs, which support both voice and text input and can return either voice or text. Included are features like APIs to support detecting when a user finishes speaking and encoding incoming audio to the format the Amazon Lex service prefers.

Amazon Lex has built-in integration with AWS Lambda to allow insertion of custom business logic into your Amazon Lex processing flow, including all of the extension to other services that Lambda makes possible.

For information on Amazon Lex concepts and service configuration, see How it Works in the Amazon Lex Developer Guide.

To get started using the Amazon Lex mobile client for iOS, you'll need to integrate the SDK for iOS into your app, set the appropriate permissions, and import the necessary libraries.

8.2 Setting Up

8.2.1 Include the SDK in Your Project

Follow the instructions on the Set Up the SDK for iOS page to include the frameworks for this service.

8.2.2 Set IAM Permissions for Amazon Lex

To use Amazon Lex in an application, create a role and attach policies as described in Step 1 of Getting Started in the Amazon Lex Developer Guide.

To learn more about IAM policies, see Using IAM.

8.2.3 Configure a Bot

To set up interaction between your mobile app and Amazon Lex, use the Amazon Lex console to configure a bot that fulfills your requirements. To learn more see Amazon Lex Developer Guide. For a quickstart, see Step 2 of Getting Started in the Amazon Lex Developer Guide.

Amazon Lex also supports model building APIs, which allow creation of bots, intents, and slots at runtime. This SDK does not currently offer additional support for interacting with Amazon Lex model building APIs.

8.3 Implement Text and Voice Interaction with Amazon Lex

8.3.1 Add Permissions and Get Credentials

Take the following steps to allow your app to access device resources and AWS services.

Add permission to use the microphone

To add permission to use the microphone to enable users to speak to Amazon Lex through your app, open your project's Info.plist file using *Right-click* > *Open As* > *Source Code*, and then add the following entry.

```
<plist version="1.0">
    . . .
    <dict>
        <key>NSMicrophoneUsageDescription</key>
        <string>For interaction with Amazon Lex</string>
    </dict>
     . . .
</plist>
```

8.3.2 Integrating the Interaction Client

Take the following steps to integrate the Amazon Lex interaction client with your app.

Initialize the *InteractionKit* for voice and text

Add the following code using the name and alias of your Lex bot to initialize an instance of *InteractionKit*.

Swift

```
let chatConfig = AWSLexInteractionKitConfig.
↪defaultInteractionKitConfig(withBotName: BotName, botAlias:␣
↪BotAlias)

// interaction kit for the voice button
AWSLexInteractionKit.register(with: configuration!,␣
↪interactionKitConfiguration: chatConfig, forKey:
↪"AWSLexVoiceButton")
```

```
chatConfig.autoPlayback = false

// interaction kit configuration for the client
AWSLexInteractionKit.register(with: configuration!,␣
↪interactionKitConfiguration: chatConfig, forKey: "chatConfig")
```

Objective C

```
AWSLexInteractionKitConfig *chatConfig =␣
↪[AWSLexInteractionKitConfig␣
↪defaultInteractionKitConfigWithBotName:BotName botAlias:
↪BotAlias];

chatConfig.autoPlayback = NO;

[AWSLexInteractionKit␣
↪registerInteractionKitWithServiceConfiguration:configuration␣
↪interactionKitConfiguration:chatConfig forKey:
↪AWSLexChatConfigIdentifierKey];
```

Implement *InteractionKit* delegate methods

Declare and implement the following methods in the class where you intend to use your *InteractionKit*:

- `interactionKit` is called to begin a conversation. When passed `interactionKit`, `switchModeInput`, and `completionSource`, the function should set the mode of interaction (audio or text input and output) and pass the `SwitchModeResponse` to the `completionSource`. On error, the *interactionKit:onError* method is called.

  Swift

  ```
  public func interactionKit(_ interactionKit:␣
  ↪AWSLexInteractionKit, switchModeInput:
    AWSLexSwitchModeInput, completionSource:␣
  ↪AWSTaskCompletionSource<AWSLexSwitchModeResponse>?)

  public func interactionKit(_ interactionKit:␣
  ↪AWSLexInteractionKit, onError error: Error)
  ```

  Objective C

  ```
  - (void)interactionKit:(AWSLexInteractionKit *)interactionKit
      switchModeInput:(AWSLexSwitchModeInput *)switchModeInput
    completionSource:(AWSTaskCompletionSource
  ↪<AWSLexSwitchModeResponse *> *)completionSource

  - (void)interactionKit:(AWSLexInteractionKit *)interactionKit
    onError:(NSError *)error`
  ```

- `interactionKitContinue` is called to continue an ongoing conversation with its transaction state and metadata maintained.

  Swift

```
func interactionKitContinue(withText interactionKit:␣
↪AWSLexInteractionKit, completionSource:␣
↪AWSTaskCompletionSource<NSString>){
    textModeSwitchingCompletion = completionSource
}
```

  Objective C

```
- (void)interactionKitContinueWithText:(AWSLexInteractionKit␣
↪*)interactionKit
    completionSource:(AWSTaskCompletionSource<NSString *>␣
↪*)completionSource{
 textModeSwitchingCompletion = completionSource;
}
```

Alternatively, you can explicitly set *SwitchModeResponse* to a selected mode.

Swift

```
let switchModeResponse = AWSLexSwitchModeResponse()
switchModeResponse.interactionMode = AWSLexInteractionMode.
↪text
switchModeResponse.sessionAttributes = switchModeInput.
↪sessionAttributes
completionSource?.setResult(switchModeResponse)
```

Objective C

```
AWSLexSwitchModeResponse *switchModeResponse =␣
↪[AWSLexSwitchModeResponse new];
[switchModeResponse setInteractionMode:
↪AWSLexInteractionModeText];
[switchModeResponse setSessionAttributes:switchModeInput.
↪sessionAttributes];
[completionSource setResult:switchModeResponse];
```

Begin or Continue a Conversation

When you call `InteractionKit` to provide input for a conversation, check if the conversation is already in progress by examining the state of `AWSTaskCompletionSource`. The following example illustrates the case where `textModeSwitchingCompletion` is an `AWSTaskCompletionSource` instance and the desired result is that a new conversation will be in the `texttInTextOut` mode.

Swift

```
if let textModeSwitchingCompletion = textModeSwitchingCompletion
↪{
        textModeSwitchingCompletion.setResult(text)
        self.textModeSwitchingCompletion = nil
    }
    else {
        self.interactionKit?.textInTextOut(text)
    }
```

Objective C

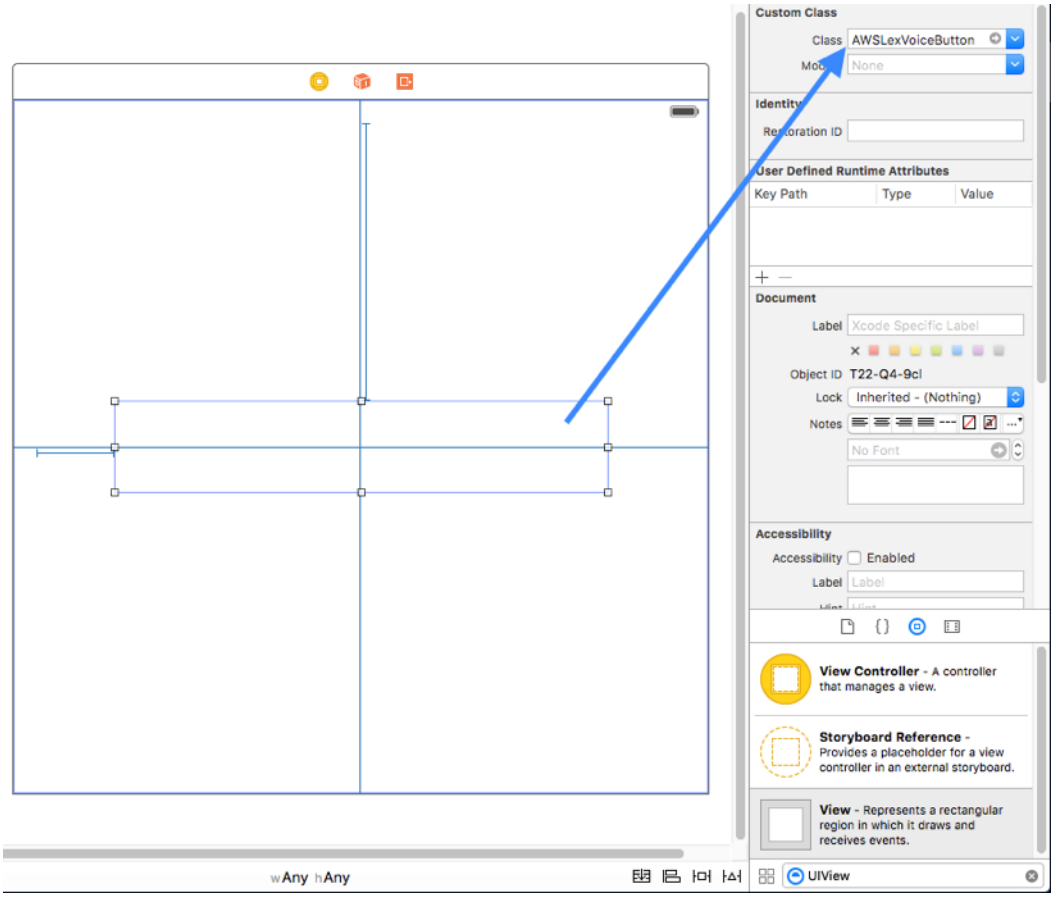
```
if(textModeSwitchingCompletion){
    [textModeSwitchingCompletion setResult:text];
    textModeSwitchingCompletion = nil;
  }else{
    [self.interactionKit textInTextOut:text];
}
```

8.3.3 Integrating Voice Conversation

Perform the following tasks to implement voice interaction with Amazon Lex in your iOS app.

Add a voice button and bind it to the Lex SDK UI component

Add a voice UIView into your storyboard scene or xib file, add a voice button (the UI element that enables users to speak to Amazon Lex). Map the voice button to the SDK button component by setting the *class* for the voice UIView to *AWSLexVoiceButton* as illustrated in the following image.


Custom Class
Class AWSLexVoiceButton
None
Identity
User Defined Runtime Attributes
Key Path
Type
Value
Document
Label
Xcode Specific Label
Object ID T22-Q4-9cl
Lock Inherited - (Nothing)
Notes
No Font
Accessibility
Accessibility Enabled
Label
View Controller - A controller that manages a view.
Storyboard Reference - Provides a placeholder for a view controller in an external storyboard.
View - Represents a rectangular region in which it draws and receives events.
wAny hAny
UIView

CHAPTER 9

Amazon Polly: Text to Speech Conversion

9.1 What is Amazon Polly?

Amazon Polly is a service that turns text into lifelike speech, making it easy to develop mobile applications that use high-quality speech to increase engagement and accessibility. With Amazon Polly you can quickly build speech-enabled apps that work in multiple geographies.

Using the following resources, you can integrate Amazon Polly with your iOS app to add text to speech transformation. No deep knowledge of either AWS services or natural language computing is needed.

For information on Amazon Polly concepts and service configuration, see How it Works in the Amazon Polly Developer Guide.

For instructions on how to integrate Amazon Polly into your iOS application, see the Application Examples section of the Amazon Polly Developer Guide.

For end to end sample apps using Amazon Polly see the AWS SDK for iOS samples.

CHAPTER 10

Amazon Pinpoint: Create Push Notification Campaigns

10.1 What is Amazon Pinpoint?

Using Amazon Pinpoint, you can create push notification campaigns that provide your users with timely, relevant, personalized information to encourage them to keep using your mobile app. You can use push notification campaigns to increase app awareness, downloads, and launches; build customer loyalty; and ultimately boost your mobile revenues.

Campaigns can include things such as welcome and onboarding messages, invites to less-engaged or lapsed users, location-based invites, time-based reminders, and special offers for frequent users. You can use Amazon Pinpoint to create and push different campaigns based on groupings of specific user characteristics, which we refer to as user segments.

Using the following resources, you can integrate Amazon Pinpoint with your iOS app to make it ready to be part of a campaign.

For information on Amazon Pinpoint concepts and service configuration, see the Amazon Pinpoint Developer Guide.

For instructions on how to integrate Amazon Pinpoint into your iOS application, see Integrating Amazon Pinpoint With iOS Apps.

CHAPTER 11

Amazon Machine Learning

Amazon Machine Learning (ML) is a service that makes it easy for developers of all skill levels to use machine learning technology. The SDK for iOS provides a simple, high-level client designed to help you interface with Amazon Machine Learning service. The client enables you to call Amazon ML's real-time API to retrieve predictions from your models and enables you to build mobile applications that request and take actions on predictions. The client also enables you to retrieve the real-time prediction endpoint URLs for your ML models.

11.1 Integrate Amazon Machine Learning

To use the Amazon Machine Learning mobile client, you'll need to integrate the SDK for iOS into your app and import the necessary libraries. To do so, follow these steps:

#. Download the SDK and unzip it as described in Setup the SDK for iOS #. The instructions direct you to import the headers for the services you'll be using. For Amazon Machine Learning, you need the following import.

Swift

```
import AWSMachineLearning
```

Objective C

```
#import <AWSMachineLearning/AWSMachineLearning.h>
```

11.1.1 Configure Credentials

You can use Amazon Cognito to provide temporary AWS credentials to your application. These credentials let the app access your AWS resources. To create a credentials provider, follow the instructions at Providing AWS Credentials.

To use Amazon Machine Learning in an application, you must set the proper permissions. The following IAM policy allows the user to perform the actions shown in this tutorial on two actions identified by ARN.

```
{
    "Statement": [{
    "Effect": "Allow",
    "Action": [
        "machinelearning:GetMLModel",
        "machinelearning:Predict"
    ],
    "Resource": "arn:aws:machinelearning:use-east-1:11122233444:
↪mlmodel/example-model-id"
    }]
}
```

This policy should be applied to roles assigned to the Amazon Cognito identity pool, but you will need to replace the Resource value with the correct account ID and ML Model ID. You can apply policies at the IAM console. To learn more about IAM policies, see Introduction to IAM.

11.1.2 Create an Amazon Machine Learning Client

Once you've imported the necessary libraries and have your credentials object, you can instantiate AWSMachineLearningGetMLModelInput.

Swift

```
let getMlModelInput = AWSMachineLearningGetMLModelInput()
```

Objective C

```
AWSMachineLearningGetMLModelInput *getMLModelInput =␣
↪[AWSMachineLearningGetMLModelInput new];
```

11.1.3 Making a Predict Request

Prior to calling Predict, make sure you have not only a completed ML Model ID but also a created real-time endpoint for that ML Model ID. This cannot be done through the mobile SDK; you will have to use the Machine Learning Console or an alternate SDK. To validate that this ML can be used for real-time Predictions.

Swift

```
// Use a created model that has a created real-time endpoint
let mlModelId = "example-model-id";
// Call GetMLModel to get the realtime endpoint URL
let getMlModelInput = AWSMachineLearningGetMLModelInput()
getMlModelInput!.mlModelId = mlModelId;

machineLearning.getMLModel(getMlModelInput!).
↪continueOnSuccessWith { (task) -> Any? in
    if let getMLModelOutput = task.result {

        if (getMLModelOutput.status !=␣
↪AWSMachineLearningEntityStatus.completed) {
```

```
                print("ML Model is not completed");
                return nil;
        }

        // Validate that the realtime endpoint is ready
        if (getMLModelOutput.endpointInfo!.endpointStatus !=␣
↪AWSMachineLearningRealtimeEndpointStatus.ready) {
            print("Realtime endpoint is not ready");
            return nil;
        }
    }

return nil
}
```

Objective C

```
// Use a created model that has a created real-time endpoint
NSString *MLModelId = @"example-model-id";

// Call GetMLModel to get the realtime endpoint URL
AWSMachineLearningGetMLModelInput *getMLModelInput =␣
↪[AWSMachineLearningGetMLModelInput new];
getMLModelInput.MLModelId = MLModelId;

[[[MachineLearning getMLModel:getMLModelInput]␣
↪continueWithSuccessBlock:^id(AWSTask *task) {
    AWSMachineLearningGetMLModelOutput *getMLModelOutput = task.
↪result;

    // Validate that the ML model is completed
    if (getMLModelOutput.status !=␣
↪AWSMachineLearningEntityStatusCompleted) {
        NSLog(@"ML Model is not completed");
        return nil;
    }

    // Validate that the realtime endpoint is ready
    if (getMLModelOutput.endpointInfo.endpointStatus !=␣
↪AWSMachineLearningRealtimeEndpointStatusReady) {
        NSLog(@"Realtime endpoint is not ready");
        return nil;
    }
}
```

Once the real-time endpoint is ready, we can begin calling Predict. Note that you must pass the real-time endpoint through the PredictRequest.

Swift

```
// Create a Predict request with your ML Model id and the␣
↪appropriate
let predictInput = AWSMachineLearningPredictInput()
```

```
predictInput!.predictEndpoint = getMLModelOutput.endpointInfo!.
 ↪endpointUrl;
predictInput!.mlModelId = mlModelId;
predictInput!.record = record

return machineLearning.predict(predictInput!)
```

Objective C

```
// Create a Predict request with your ML Model id and the␣
 ↪appropriate Record mapping.
AWSMachineLearningPredictInput *predictInput =␣
 ↪[AWSMachineLearningPredictInput new];
predictInput.predictEndpoint = getMLModelOutput.endpointInfo.
 ↪endpointUrl;
predictInput.MLModelId = MLModelId;
predictInput.record = @{};

// Call and return prediction
return [MachineLearning predict:predictInput];
```

Additional Resources

- Developer Guide
- API Reference

CHAPTER 12

Amazon Mobile Analytics: Try Amazon Pinpoint

Note: Amazon Mobile Analytics is now part of a more fully featured service called Amazon Pinpoint. Using Amazon Pinpoint, you can gather and view customized metrics, as in Amazon Mobile Analytics, and then create a campaign to respond to your user's behavior via push notification based on the data you receive.

Existing Amazon Mobile Analytics for your mobile apps are automatically ported to the Amazon Pinpoint console and will continue to function as expected.

Using Amazon Mobile Analytics, you can track customer behaviors, aggregate metrics, generate data visualizations, and identify meaningful patterns. The AWS SDK for iOS provides integration with the Amazon Mobile Analytics service. For information about AWS service region availability or Mobile Analytics region availability, see AWS Service Region Availability.

12.1 What is Amazon Mobile Analytics?

Amazon Mobile Analytics lets you collect, visualize, and understand app usage for your iOS, Android, FireOS, Windows Phone, Blackberry mobile apps as well as desktop and web apps running on Windows, OS X, and Linux. Reports are available for metrics on active users, sessions, retention, in-app revenue, and custom events, and can be filtered by platform and date range. Amazon Mobile Analytics is built to scale with your business and can collect and process billions of events from many millions of endpoints.

To learn more about Mobile Analytics, see the Amazon Mobile Analytics User Guide

12.1.1 IAM Policy for Amazon Mobile Analytics

To use Amazon Mobile Analytics, AWS users must have the correct permissions. The following IAM policy allows the user to submit events to Amazon Mobile Analytics.

```
{
    "Statement": [{
        "Effect": "Allow",
        "Action": "mobileanalytics:PutEvents",
```

```
        "Resource": "*"
    }]
}
```

This policy should be assigned to roles associated with the Amazon Cognito identity pool for your app. The policy allows clients to record events with the Mobile Analytics service. Amazon Cognito will set this policy for you, if you let it create new roles. Other policies are required to allow IAM users to view reports.

You can set permissions at https://console.aws.amazon.com/iam/. To learn more about IAM policies, see Using IAM.

12.2 Integrating Amazon Mobile Analytics

The sections below explain how to integrate Amazon Mobile Analytics with your app.

12.2.1 Create an App in the Amazon Mobile Analytics Console

Go to the Amazon Mobile Analytics console and create an app. Note the `appId` value, as you'll need it later.

To learn more about creating new apps in the console, see Managing Apps in the Amazon Mobile Analytics User Guide.

12.2.2 Integrate the SDK into Your App

If you haven't already done so, download the SDK for iOS, unzip it, and include it in your application as described at *Set Up the SDK for iOS*. The instructions direct you to import the headers for the services you'll be using.

Objective-C

```
#import <AWSMobileAnalytics/AWSMobileAnalytics.h>
```

Initialize an **`AWSMobileAnalytics`** client. In doing so, you'll need to provide the `appId` value that you generated in the Amazon Mobile Analytics console. The `appId` is used to group your data in the Amazon Mobile Analytics console.

Objective-C

```
AWSMobileAnalyticsConfiguration *analyticsConfiguration =␣
↪[[AWSMobileAnalyticsConfiguration alloc] init];
[analyticsConfiguration setServiceConfiguration:
↪serviceConfiguration];
AWSMobileAnalytics *analytics = [AWSMobileAnalytics␣
↪mobileAnalyticsForAppId:@"yourAppId" configuration:␣
↪analyticsConfiguration];
```

where "yourAppId" is the `appId` value from the Amazon Mobile Analytics console.

12.2.3 Add Monetization Events

The SDK for iOS provides the `AWSMobileAnalyticsAppleMonetizationEventBuilder` class, which helps you build monetization events to track purchases from Apple's IAP Framework.

To learn more about monetization events, see:

- AWSMobileAnalyticsAppleMonetizationEventBuilder in the API Reference Guide.
- Creating Monetization Events in the Amazon Mobile Analytics User Guide.

12.2.4 Record Custom Events

To record custom events, we first need to get the event client from the AWSMobileAnalytics instance.

Objective-C

```
id<AWSMobileAnalyticsEventClient> eventClient = analytics.
↪eventClient;
```

For this example, let's say your app is a game, and you want to record an event when a user completes a level. Create a "LevelComplete" event.

Objective-C

```
id<AWSMobileAnalyticsEvent> levelEvent = [eventClient␣
↪createEventWithEventType:@"LevelComplete"];
```

Note that custom events can't start with an underscore (_), or they'll be filtered out.

Add attributes and metrics to the event in key-value pairs.

Objective-C

```
[levelEvent addAttribute:@"Upper Dungeon" forKey:@"LevelName"];
[levelEvent addAttribute:@"Moderately difficult" forKey:@
↪"Difficulty"];
[levelEvent addMetric:@1763 forKey:@"TimeToComplete"];
```

Record the event.

Objective-C

```
[eventClient recordEvent:levelEvent];
```

Events are submitted automatically when the user goes into the background. However, if you want to submit events manually, you can do so with the **`submitEvents`** method.

Objective-C

```
[eventClient submitEvents];
```

If you don't call **`submitEvents`**, events will automatically be submitted at periodic intervals.

To learn more about custom events, see:

- AWSMobileAnalyticsEventClient in the API Reference Guide.
- AWSMobileAnalyticsEvent in the API Reference Guide.
- Creating a Custom Event in the Amazon Mobile Analytics User Guide.

CHAPTER 13

Additional Resources

- For more information about the AWS SDK for iOS, including a complete list of supported AWS products, see the AWS Mobile SDK product page.
- The SDK reference documentation includes the ability to browse and search code included with the SDK. It provides thorough documentation and usage examples. You can find it at AWS SDK for iOS API Reference.
- Post questions and feedback at the Mobile Developer Forum.
- Source code and sample applications are available at AWS SDK for iOS repository. Source code for Amazon Cognito is available at the Amazon Cognito repository.