AWS SDK for iOS 2.0

Developer Guide

AWS SDK for iOS 2.0: Developer Guide

Copyright © 2014 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

The following are trademarks of Amazon Web Services, Inc.: Amazon, Amazon Web Services Design, AWS, Amazon CloudFront, Cloudfront, Amazon DevPay, DynamoDB, ElastiCache, Amazon EC2, Amazon Elastic Compute Cloud, Amazon Glacier, Kindle, Kindle Fire, AWS Marketplace Design, Mechanical Turk, Amazon Redshift, Amazon Route 53, Amazon S3, Amazon VPC. In addition, Amazon.com graphics, logos, page headers, button icons, scripts, and service names are trademarks, or trade dress of Amazon in the U.S. and/or other countries. Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon.

All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

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.

Note
This guide describes AWS SDK for iOS 2.0, which is currently in developer preview.

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.

Amazon Cognito

Amazon Cognito facilitates the delivery of scoped, temporary credentials to mobile devices or other un-trusted environments, and it uniquely identifies a device or user and supplies the user with a consistent identity throughout the lifetime of an application. The 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 How to Authenticate Users (p. 8) and How to Sync Profile Data (p. 15).

Amazon Mobile Analytics

Amazon Mobile Analytics 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.

After you've set up the SDK, you can start using Amazon Mobile Analytics by following the instructions at How to Use Analytics in Your App (p. 18).

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/**
  - **AWSiOSSDKv2.framework**
  - **extras/**
    - **AWSCognitoSync.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.
  - **third-party/**–Contains third-party frameworks.

Next Steps

- To start using the SDK for iOS, follow the setup instructions at Set Up the SDK for iOS (p. 3).
- For more information about the AWS SDK for iOS, including a complete list of supported AWS products, go to http://aws.amazon.com/mobile/sdk.
- 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 project on GitHub.

Set Up the SDK for iOS

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

Setting Up

To get started, you will need to get the SDK for iOS and set up a credential provider using Amazon Cognito.

Get the iOS SDK

To use the AWS SDK for iOS, you will need the following installed on your development machine:

- Xcode 5 or later
- iOS 7 or later

Download the SDK from http://aws.amazon.com/mobile/sdk. The SDK is stored in a compressed file archive named `aws-ios-sdk-#.#.#.zip` (where '#.#.#' represents the version number, so for version 2.0.0, the filename is `aws-ios-sdk-2.0.0`).

You can also download the SDK source code.

Include the SDK for iOS in an Existing Application

The samples included with the SDK for iOS are standalone projects that are already set up for you. You can also integrate the SDK for iOS with an existing application. If you have an existing app in which you'd like to use AWS, follow these steps:

1. With your project open in Xcode, Control+click **Frameworks** and then click **Add files to "<project name>"...**.

2. In Finder, navigate to the `AWSiOSSDKv2.framework` file and select it. Click **Add**. If you want to use Amazon Cognito Sync, you also need to add the `AWSCognitoSync.framework`, which is in the `extras` directory.
3. In your source code, import the appropriate headers for the services you are using. The header file import convention is FRAMEWORKNAME/SERVICENAME.h, as in the following example:

   ```
   #import <AWSiOSSDKv2/S3.h>
   #import <AWSiOSSDKv2/DynamoDB.h>
   #import <AWSiOSSDKv2/SQS.h>
   #import <AWSiOSSDKv2/SNS.h>
   ```
4. Open a target for your project, select **Build Phases**, expand **Link Binary With Libraries**, click the **+** button, and add `libsqlite3.dylib` and `libz.dylib`.
5. In the target for your project, select **Build Settings**, go to **Linking** and ensure that it's expanded. Add the `-ObjC` linker flag to **Other Linker Flags**.
6. If your project does not already include them, drag the following frameworks, located in the `third-party` directory, into your project.

   - `Bolts.framework` (If your application uses the Facebook SDK, you won't need this framework, as it's already included with the Facebook SDK.)
   - `GZIP.framework`
   - `Mantle.framework`
   - `Reachability.framework`
   - `TMCache.framework`
   - `UICKeyChainStore.framework`
   - `XMLDictionary.framework`

7. Drag and drop the following JSON files, located in the `service-definitions` directory, into your project.

   - `autoscaling-2011-01-01.json`
   - `cib-2014-06-30.json`
   - `css-2014-06-30.json`
   - `dynamodb-2012-08-10.json`
   - `ec2-2014-06-15.json`
   - `elasticloadbalancing-2012-06-01.json`
   - `email-2010-12-01.json`
   - `kinesis-2013-12-02.json`
   - `mobileanalytics-2014-06-30.json`
   - `monitoring-2010-08-01.json`
   - `s3-2006-03-01.json`
   - `sdb-2009-04-15.json`
   - `sns-2010-03-31.json`
   - `sqs-2012-11-05.json`
   - `sts-2011-06-15.json`

Note
You can change the log level by importing `AWSCore.h` and calling:

```
[AWSLogger defaultLogger].logLevel = AWSLogLevelVerbose;
```

The following logging level options are available:

- `AWSLogLevelNone`
- `AWSLogLevelError` (This is the default. Only error logs are printed to the console.)
- `AWSLogLevelWarn`
- `AWSLogLevelInfo`
- `AWSLogLevelDebug`
- `AWSLogLevelVerbose`

Get AWS Credentials with Amazon Cognito or AWS Identity and Access Management

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 How to Authenticate Users (p. 8).

Alternatively, you can use AWS Identity and Access Management (IAM). If you choose IAM, ensure that your role's policy is minimally scoped so that it can only perform the desired actions for the service being used.

Run the Demos

The AWS SDK for iOS includes sample apps that demonstrate common use cases. To import dependencies, you can either add frameworks, as described above, or you can use CocoaPods. If you use CocoaPods, you'll need a podfile in your application folder, to which you'll add dependencies.

For example, to use the iOS SDK and Amazon Cognito Sync, you'd add the following to your podfile:

```
pod 'AWSiOSSDKv2'
pod 'AWSCognitoSync'
```

Then you would run the **pod install** command. CocoaPods will generate a workspace for you.

Run the Amazon S3 `TransferManager` Sample

This sample demonstrates the Amazon S3 `TransferManager` found in the AWS SDK for iOS.

1. Create an Amazon S3 bucket. (For details on creating a bucket in the Amazon S3 console, see Create a Bucket.)
2. In the Amazon Cognito console, create an identity pool and download or copy the starter code snippets. Make sure the role has full permissions for the bucket you created.
3. Open `S3TransferManagerSample.xcodeproj`.
4. Open `Constants.m` and update the following lines with the appropriate constants:

```
NSString *const AWSAccountID = @"Your-AccountID";
NSString *const CognitoPoolID = @"Your-PoolID";
NSString *const CognitoRoleUnauth = @"Your-RoleUnauth";
NSString *const S3BucketName = @"Your-S3-Bucket-Name";
```

5. Build and run the sample app.

Run the Amazon DynamoDB Object Mapper Sample

This sample demonstrates the DynamoDB object mapper found in the AWS SDK for iOS.

1. In the Amazon Cognito console, use Amazon Cognito to create a new identity pool. Obtain the `AccountID`, `PoolID`, and `RoleUnauth` constants. Make sure the role has full permissions for the sample table.
2. Open `DynamoDBSample.xcodeproj`.
3. Open `Constants.m` and update the following lines with the constants from step 1:
   - `NSString *const AWSAccountID = @"Your-AccountID";`
   - `NSString *const CognitoPoolID = @"Your-PoolID";`
   - `NSString *const CognitoRoleUnauth = @"Your-RoleUnauth";`
4. Build and run the sample app.

Install the Reference Documentation in Xcode

The AWS 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 Mac OS X terminal.

To install the DocSet for Xcode

1. Open the Mac OS X terminal and go to the directory containing the expanded archive. For example:

```
$ cd ~/Downloads/aws-ios-sdk-2.0.0
```

Note
Remember to replace `2.0.0` in the example above with the actual version number of the AWS SDK for iOS that you downloaded.

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

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

3. 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/Doc
umentation/DocSets/
```

4. If Xcode was running during this procedure, restart Xcode. To browse the documentation, go to **Help**, click **Documentation and API Reference**, and select **AWS SDK for iOS v#.# Documentation**.

Getting Started with Services

Using the AWS SDK for iOS, you can build iOS applications that rely on AWS for user authentication, data syncing, analytics, and other services. The following topics show you how to get started.

Topics

How to Authenticate Users

Using Amazon Cognito, you can create unique identities for your users and authenticate them for secure access to your AWS resources. Amazon Cognito also provides a cloud sync store and synchronization APIs for end user profiles. With Amazon Cognito, you can support public identity providers such as Amazon, Facebook, and Google, as well as unauthenticated identities. To use Amazon Cognito in your app, you must include `AWSCognitoSync.framework` in your project.

Topics

Amazon Cognito Overview

Amazon Cognito is a web service that simplifies the process of authenticating mobile users and syncing user data across mobile devices. Amazon Cognito comprises two services, each with its own API.

The Amazon Cognito Identity service is intended to ease the delivery of scoped, temporary credentials to mobile devices or other untrusted environments. It works in concert with AWS Identity and Access Management (IAM) and AWS Security Token Service (AWS STS) to uniquely identify a user and supply the user with a consistent identity throughout the lifetime of an application. The following sections focus on Amazon Cognito Identity.

Amazon Cognito Sync is a synchronization service designed to enable cross-device syncing of application data. A high-level client library, available for both Android and iOS, stores the data locally so it's available even if the device is offline. To learn more about using Cognito Sync in an application, see How to Sync Profile Data (p. 15)

When you use Amazon Cognito, you decide if users can run your app without authenticating or if they have to sign in using one of the supported identity providers: Facebook, Google, or Amazon. User data that's saved when the user is running unauthenticated is preserved when the user signs in.

Prerequisites

To integrate Amazon Cognito with your app, you need the following:

- An AWS account
- An Xcode development environment
- AWS Mobile SDK for iOS
- Optionally, a developer account and an application registered with the identity provider you want to use (Facebook, Google, or Amazon)

Cognito Security

When application users log in, Amazon Cognito generates a unique identifier and delivers temporary AWS credentials. These credentials are tied to a user identity and are associated with a specific IAM role.

When creating an Amazon Cognito identity pool, you are asked to specify a role (or create a new one) that authenticated and unauthenticated users will assume when logged in to an AWS account. The role identifiers (ARNs) are passed to the Amazon Cognito APIs from the client device.

The default roles created by the Amazon Cognito management console give end users access to certain AWS services. These include the following:

- Amazon Cognito sync service
- Mobile Analytics service

To allow or restrict access to other services, log in to the IAM management console, click **Roles**, and select a role generated for the Amazon Cognito identity pool. The policies attached to the selected role are listed in the **Permissions** tab. You can customize an access policy by clicking the corresponding **Manage Policy** link.

To learn more about using and defining policies, see the Overview of IAM Policies in the *AWS Identity and Access Management User Guide*.

For Amazon Cognito to work, the IAM policy must at least enable access to the Amazon Cognito sync store for each identity, as in the following example:

```
{
  "Version": "2012-10-17",
  "Statement":[{
    "Effect":"Allow",
    "Action":"cognito-sync:*",
   "Resource":["arn:aws:cognito-sync:us-east-1:123456789012:identitypool/${cog
nito-identity.amazonaws.com:aud}/identity/${cognito-identity.amazon
aws.com:sub}/*"]
  }]
}
```

If you want access to your entire identity pool, you can use the following snippet to give your IAM user access to the entire Amazon Cognito sync store:

```
{
  "Version": "2012-10-17",
  "Statement":[{
    "Effect":"Allow",
    "Action":"cognito-sync:*",
    "Resource":["arn:aws:cognito:us-east-1:123456789012:identitypool/*"]
  }]
}
```

Amazon Cognito supports both unauthenticated and authenticated users. Authenticated users are authenticated by one of three public login providers: Amazon, Facebook, or Google. The two different identity types—unauthenticated and authenticated—should be assigned separate IAM roles and policies. When you create a new identity pool in the Cognito console, Cognito creates the roles and policies for you.

Cognito Identity Pools

Before you integrate with Amazon Cognito, you'll need to create an identity pool where your app can store data. The identity pool is a store of user identity information specific to your account. The information is retrievable across client platforms, devices, and operating systems, so that if a user starts using the app on a phone and later switches to a tablet, the persisted app information is still available for that user.

To create a new identity pool for your users, log in to the Amazon Cognito console. The **New Identity Pool** wizard will guide you through the creation of your first identity pool.

When an identity pool is created successfully, you'll have access to the AWS account ID, the new Amazon Cognito identity pool ID, and the role ARNs necessary to initialize the Amazon Cognito client.

Setting Up Identity Provider Applications

Amazon Cognito integrates with Facebook, Google, and Amazon to provide federated authentication for your mobile application users. In order to verify tokens from a given identity provider, Amazon Cognito needs to know the identifier of the application that you registered with the provider. Amazon Cognito asks for identifiers during the identity pool creation process.

This section explains how to register and set up your application with each identity provider.

Facebook

You need to register your application with Facebook before you can start authenticating Facebook users and interacting with Facebook APIs.

The Facebook Developers portal takes you through the process of declaring and setting up your application.

1. At the Facebook Developers portal, log in with your Facebook credentials.
2. From the Apps menu select **Create a New App**.
3. In the create application window, specify the **Display Name** and **Namespace** for your application. Once your application is created, Facebook will take you to the application dashboard.

The top section of the application dashboard shows you the **App ID** and **App Secret** that Facebook has generated for your application. You will need the application ID to set up your Amazon Cognito Identity Pool for Facebook login.

Both the iOS and Android guides on the Facebook developers portal have a comprehensive getting started guide detailing how to integrate your application with Facebook Login.

Google

To enable Google+ Sign-in for iOS and Android, you will need to create a Google Developers console project for your application.

1. Go to the Google Developers Console and log in with your Google account.
2. Click the **Create Project** button and give your project a name and an ID.
3. Once the project is created, select **APIS & AUTH** and then **APIs** to open the APIs screen.
4. Ensure the **Google+ API** status is set to ON.
5. Select **Credentials**.
6. Click **Create New Client ID** in the **OAuth** credentials category.
7. In the form that appears, select **Installed application** and then your platform of choice. The form will ask you for your application details.

You will need to use the generated client ID to authenticate users in your application. You will also need to specify your Google client ID when creating your Amazon Cognito identity pool.

To learn more, see the comprehensive Google documentation for iOS or Android.

Amazon

To enable Login with Amazon, you'll need to create an Application ID in the Amazon App Console.

1. Navigate to the Amazon App Console and login with your Amazon account.
2. Click **Register New Application**. The form will ask you for a name, description, privacy note, and logo for your application. The next screen will show your Amazon App ID.
3. Open the iOS or Android settings and configure your application to allow login with Amazon.

You will need to specify the generated app ID when creating your Amazon Cognito identity pool.

The Amazon Developer portal has a comprehensive guide on how to integrate Login with Amazon in your iOS and Android applications.

Integrating Identity Providers

You can use the `logins` property to set up credentials received from an identity provider. The instructions below guide you through authentication with the identity providers supported by Amazon Cognito.

You can associate a Amazon Cognito identity with multiple identity providers by setting all required tokens in the `logins` property. For example, you could set both the Facebook and Google tokens in the `logins` property, so that the unique Amazon Cognito identity would be associated with both identity provider logins. No matter which account the end user uses for authentication, Amazon Cognito returns the same user identifier.

Facebook

To provide Facebook authentication, first follow the Facebook guide to include their SDK in your application. Then add a "Login with Facebook" button to your iOS user interface. The Facebook SDK uses a session object to track its state. The access token property of the session object is what Amazon Cognito uses to authenticate the user, generate the unique identifier, and, if needed, grant the user access to other AWS resources.

Once you have authenticated your end user with the Facebook SDK, add the session token to the Amazon Cognito credentials provider.

```
NSString *token = FBSession.activeSession.accessTokenData.accessToken;
credentialsProvider.logins = @{ AWSCognitoLoginProviderKeyFacebook: token };
```

The Facebook `logins` process initializes a singleton session within its SDK. The Facebook session object contains an Open ID token that Amazon Cognito uses to generate AWS credentials for your authenticated end user. Amazon Cognito also uses the token to check against your user database for the existence of a user matching this particular Facebook identity. If the user already exists, the existing identifier is returned by the APIs. Otherwise, a new identifier is returned. Identifiers are automatically cached by the client SDK on the local device.

Google

To enable login with Google in your application, follow the Google+ documentation for iOS. A successful authentication with Google results in a `GTMOAuth2Authentication` token. Amazon Cognito uses this token to authenticate the user and generate the unique identifier.

```
- (void)finishedWithAuth: (GTMOAuth2Authentication *)auth error: (NSError *)
error {
    NSString *idToken = [auth.parameters objectForKey:@"id_token"];
   credentialsProvider.logins = @{ AWSCognitoLoginProviderKeyFacebook: idToken
 };
}
```

Amazon

The login with Amazon guide takes you through the steps necessary to set up Login with Amazon in your application, download the client SDK, and declare your application on the Amazon developer platform.

Once login with Amazon is implemented, you can pass the token to the Cognito credentials provider in the `requestDidSucceed` method of the `AMZNAccessTokenDelegate`.

```
- (void)requestDidSucceed:(APIResult *)apiResult {
    credentialsProvider.logins = @{ AWSCognitoLoginProviderKeyLoginWithAmazon:
 apiResult.result };
}
```

Providing AWS Credentials

Amazon Cognito generates unique identifiers for your users and acts as a credentials provider to AWS services. For each user, Amazon Cognito delivers a temporary access token retrieved from the AWS Security Token Service.

By default, the credentials provider assumes it is dealing with an unauthenticated identity. If your application logs in with an identity provider, you'll need to add the login method to the credentials provider.

To provide AWS credentials to your app:

1. In the Amazon Cognito console, create an identity pool and download or copy the starter code snippets.
2. If you haven't already done so, add the `AWSiOSSDKv2.framework` to your project. (For more information, see Include the SDK for iOS in an Existing Application (p. 3)).
3. In your source code, include the `AWSCore` header.

   ```
   #import <AWSiOSSDKv2/AWSCore.h>
   ```

   (If you are using CocoaPods, replace `<AWSiOSSDK/AWSCore.h>` with `"AWSCore.h"`.)
4. Initialize the Amazon Cognito credentials provider using the code snippet generated by the Cognito console. The replaceable values shown below will be specific to your account:

   ```
   AWSCognitoCredentialsProvider *credentialsProvider =
      [AWSCognitoCredentialsProvider credentialsWithRegionType:AWSRegionUSEast1

                                                     accountId:@"XXXXXXXXXXXX"

                                                identityPoolId:@"us-east-
   1:XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXX"
                                                        unau
   thRoleArn:@"arn:aws:iam::XXXXXXXXXX:role/YourUnauthRoleName"
                                                          au
   thRoleArn:@"arn:aws:iam::XXXXXXXXXX:role/YourAuthRoleName"];

   AWSServiceConfiguration *configuration = [AWSServiceConfiguration configur
   ationWithRegion:AWSRegionUSEast1
                                                                        cre
   dentialsProvider:credentialsProvider];

   [AWSServiceManager defaultServiceManager].defaultServiceConfiguration =
   configuration;
   ```

Getting an Amazon Cognito ID and AWS Credentials

Once the login tokens are set up in the credentials provider object, you can retrieve the Amazon Cognito unique identifier for your end user and the temporary credentials that let the app access your AWS resources.

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

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

Once the Amazon Cognito credentials provider is initialized, you can use it as the credentials provider when you create clients for other Amazon Web Services using the mobile SDK.

```
// create a configuration that uses the provider
AWSServiceConfiguration *configuration = [AWSServiceConfiguration configuration
WithRegion:AWSRegionUSEast1
provider:credentialsProvider];
// get a client with the specified config
AWSDynamoDB *dynamoDB = [[AWSDynamoDB new] initWithConfiguration:configuration];
```

The credentials provider object communicates with the Amazon Cognito APIs and retrieves both the unique identifier for authenticated and unauthenticated users as well as temporary, limited privilege AWS credentials for the AWS Mobile SDK. The credentials retrieved by the Amazon Cognito credentials provider are valid for one hour.

Switching Identities

Users can begin their life in an application as unauthenticated guests. Eventually they may decide to log in using one of the supported identity providers. Amazon Cognito will ensure the authenticated identity retains the same unique identifier as the new, authenticated one and that the profile data is merged automatically.

The `NSNotificationCenter` informs your application of a profile merge.

```
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(iden
tityIdDidChange:) name:AWSCognitoIdentityIdChangedNotification object:nil];
-(void)identityDidChange:(NSNotification*)notification {
    NSDictionary *userInfo = notification.userInfo;
   NSLog(@"identity changed from %@ to %@", [userInfo objectForKey:AWSCognitoNo
tificationPreviousId], [userInfo objectForKey:AWSCognitoNotificationNewId]);
}
```

How to Sync Profile Data

Amazon Cognito is an AWS service and client library designed to enable cross-device syncing of application-related user data. You can use Cognito Sync to synchronize user profile data across devices and login providers such as Amazon, Facebook, and Google. To use Amazon Cognito in your app, you must include `AWSCognitoSync.framework` in your project.

Topics

Profile Data Synchronization

Amazon Cognito lets you save key-value pairs to a profile. To keep this data in sync between the service and an end user's devices, invoke the `synchronize` method. Data is stored within data sets associated with an identity. Each data set can have a maximum size of 1 MB. The total maximum size for an identity is 20 MB.

The Amazon Cognito sync client creates a local cache in a SQLite database for the identity data. When the `synchronize` method is called, changes from the service are pulled to the device and any local changes are pushed to the service.

Initializing the Cognito Sync Client

To initialize the Amazon Cognito sync client, you first need to obtain credentials. With Amazon Cognito, each identity has access only to its own data in the sync store. You have 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 (p. 13).

Once you have an Amazon Cognito credentials provider, you can initialize the Amazon Cognito sync client to write data to your end user's identity. The constructor method informs the Cognito Sync service to initialize a sync store for the identity contained in the credentials provider.

The following code shows how to initialize a sync client. The Amazon Cognito sync client reads the unique user identifier from the Amazon Cognito credentials provider embedded in the configuration object that was created in the previous example.

To initialize a sync client, you need to include the required header file and provide the Amazon Cognito credentials provider object.

```
#import <AWSiOSSDKv2/AWSCore.h>
#import <AWSCognitoSync/Cognito.h>
AWSCognito *syncClient = [AWSCognito defaultCognito];
```

(If you are using CocoaPods, replace `<AWSiOSSDKv2/AWSCore.h>` with `"AWSCore.h"`.)

Datasets

Within a Amazon Cognito identity, end user profile data is organized in datasets. Each dataset can contain up to 1 MB of data in the form of key-value pairs. A dataset is the most granular entity on which you can perform a synchronize operation.

Read and write operations performed on a dataset only affect the local store until the sync method is invoked. Datasets are identified by a unique name in the form of an alphanumeric string.

The Amazon Cognito sync client exposes a method to create a new dataset or open an existing one.

```
AWSCognitoDataset *dataset = [syncClient openOrCreateDataset:@"myDataSet"];
```

An entire dataset can be deleted from the local storage using the `clear` method of the `AWSCognitoDataSet` object. In order to delete a dataset from Amazon Cognito, call the `synchronize` method.

```
[dataset clear];
[dataset synchronize];
```

Reading and Writing Data in Datasets

Amazon Cognito datasets can be treated as dictionaries. Values written to a dataset only affect the local cached copy of the data until the synchronize method is called.

Keys and values are read, added, or modified in a dataset exactly as if it were a dictionary.

```
[dataset setString:@"my value" forKey:@"myKey"];
NSString *value = [dataset stringForKey:@"myKey"];
```

Keys can also be deleted from a dataset with the `removeObjectForKey` method.

```
[dataset removeObjectForKey:@"myKey"];
```

Synchronization

The synchronize method compares local cached data to the data stored in the Cognito sync store. Remote changes are pulled from the Cognito sync store, and then updated values on the device are pushed to the service.

A synchronize operation is initiated simply by calling the `synchronize` method of the dataset object. The `synchronize` method is asynchronous and returns a `BFTask` object to handle the response.

```
[[dataset synchronize] continueWithBlock:^id(BFTask *task) {
```

```
    if (task.isCancelled) {
        // task cancelled
    } else if (task.error) {
        // error while executing task
    } else {
        // task succeeded. The data was saved in the sync store
    }
    return nil;
}];
```

The `synchronizeOnConnectivity` method attempts to synchronize when the device has connectivity. First, `synchronizeOnConnectivity` checks for connectivity and, if the device is online, immediately invokes `synchronize` and returns the `BFTask` associated with the attempt.

If the device is offline, `synchronizeOnConnectivity` schedules a synchronize for the next time the device comes online and returns a `BFTask` with a `nil` result. The scheduled synchronize is only valid for the lifecycle of the dataset object. The data will not be synchronized if the app is exited before connectivity is regained. If you want to be notified when events occur during the scheduled synchronize, you must add observers of the notifications found in `AWSCognito`.

Conflict Resolution Handler

Applications can receive notifications regarding changes to datasets and synchronization operations by implementing `AWSCognitoRecordConflictHandler`. By using the handler, applications can make active decisions about what data is synchronized.

```
client.conflictHandler = ^AWSCognitoResolvedConflict* (NSString *datasetName,
AWSCognitoConflict *conflict) {
        // always choose local changes
        return [[AWSCognitoResolvedConflict alloc] initWithLocalRecord: con
flict];
    };
```

The conflict resolution handler can be set on the Cognito client and overridden on individual `AWSCognitoDataSet` objects.

```
dataset.conflictHandler = ^AWSCognitoResolvedConflict* (NSString *datasetName,
 AWSCognitoConflict *conflict) {
        // override and always choose remote changes
        return [[AWSCognitoResolvedConflict alloc] initWithRemoteRecord: con
flict];
    };
```

Conflicts may arise during the sync operation if the same key on the local store and cloud sync store has been modified. If the handler is not set, Cognito's default conflict resolution chooses the most recent update.

The conflict resolution method receives an `AWSCognitoRecord` object for both the local cached data and the conflicting record from the cloud sync store. The `AWSCognitoRecord` contains the key and value set for the conflicting key-value pair.

Returning `nil` from this method prevents the synchronization from continuing. The conflicts will be presented again the next time the synchronization process starts.

How to Use Analytics in Your App

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 Mobile Analytics service.

What is Amazon Mobile Analytics

Amazon Mobile Analytics lets you collect, visualize, and understand app usage for your iOS, Android, and Fire OS 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 Mobile Analytics is built to scale with your business and can collect and process billions of events from many millions of endpoints.

IAM Policy for Amazon Mobile Analytics

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

```
{
"Statement": [{
    "Effect": "Allow",
    "Action": "mobileanalytics:PutEvents",
    "Resource": "*"
}]
}
```

This policy should be applied to roles to which the Cognito identity pool is assigned. 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.

Integrating Amazon Mobile Analytics

1. If you haven't already done so, download the SDK for iOS, unzip it, and include it in your application as described at Include the SDK for iOS in an Existing Application (p. 3). The instructions direct you to import the headers for the services you'll be using. For this example, you need the following imports:

   ```
   #import <AWSiOSSDKv2/AWSCore.h>
   #import <AWSiOSSDKv2/AWSMobileAnalytics.h>
   ```

2. 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 (p. 13).

3. Initialize an `AWSMobileAnalytics` client. In doing so, you'll need to provide a globally unique ID string that differentiates your client from all other apps using Mobile Analytics.

```
 AWSMobileAnalytics* analytics = [AWSMobileAnalytics mobileAnalyticsForAp
pId:@"85e55ef0-0df7-11e4-a2a9-b2227EXAMPLE"];
```

   This example shows an altered UUID, but you could pass in the app name or some other string instead. However, it's important that the string you pass in doesn't collide with another instance of Mobile Analytics. Using a UUID minimizes the possibility of collision, and using a UUID concatenated with your app name (for example `85e55ef0-0df7-11e4-a2a9-b2227EXAMPLE-my-app`) is even safer.
4. Get the event client from the AWSMobileAnalytics instance.

```
id<AWSMobileAnalyticsEventClient> eventClient = analytics.eventClient;
```

5. 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.

```
id<AWSMobileAnalyticsEvent> levelEvent = [eventClient createEventWithEvent
Type:@"LevelComplete"];
```

   Note that custom events can't start with an underscore (_), or they'll be filtered out.
6. Add attributes and metrics to the event in key-value pairs.

```
[levelEvent addAttribute:@"Upper Dungeon" forKey:@"LevelName"];
[levelEvent addAttribute:@"Moderately difficult" forKey:@"Difficulty"];
[levelEvent addMetric:@1763 forKey:@"TimeToComplete"];
```

7. Record the event.

```
[eventClient recordEvent:levelEvent];
```

8. 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:

```
[eventClient submitEvents];
```

   If you don't call `submitEvents`, events will automatically be submitted at periodic intervals.

How to Upload Files from Your App to the Cloud

The Amazon S3 `TransferManager` simplifies the process of transferring files to and from Amazon Simple Storage Service (S3). Using the Amazon S3 `TransferManager`, you can transfer files from your app on the device to storage in the cloud.

What Is the Amazon S3 TransferManager?

The Amazon S3 TransferManager is a high-level utility for managing transfers to Amazon S3. TransferManager provides a simple API for uploading content to Amazon S3, and makes use of Amazon S3 multipart uploads to achieve enhanced throughput, performance, and reliability. TransferManager also supports progress updates, pause, and resume. When possible, TransferManager attempts to use multiple threads to upload multiple parts of a single upload at once. When dealing with large content sizes and high bandwidth, this can have a significant increase on throughput.

Uploading a File

To use Amazon S3 `TransferManager`, you'll need to configure a service client and provide AWS credentials. You can use Amazon Cognito to authenticate users and provide credentials to AWS services. 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 (p. 13).

> **Note**
> By default, the Cognito console will not create roles with permissions to S3. You will need to modify your roles in the AWS Identity and Access Management console.

Now that we have a credentials provider, let's look at an example of uploading a file using the `S3TransferManager` class. We'll also try canceling, pausing, and resuming the upload. We'll put this logic in a class called `FirstViewController`.

First, we'll create a class interface that defines four buttons and four corresponding methods.

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

@interface FirstViewController : UIViewController

@property (nonatomic, weak) IBOutlet UILabel *uploadStatusLabel;
@property (nonatomic, weak) IBOutlet UIButton *uploadButton;
@property (nonatomic, weak) IBOutlet UIButton *pauseButton;
@property (nonatomic, weak) IBOutlet UIButton *resumeButton;
@property (nonatomic, weak) IBOutlet UIButton *cancelButton;

- (IBAction)uploadButtonPressed:(id)sender;
- (IBAction)pauseButtonPressed:(id)sender;
- (IBAction)resumeButtonPressed:(id)sender;
- (IBAction)cancelButtonPressed:(id)sender;

@end
```

Then, in the class implementation, we'll use a `viewDidLoad` method to create a new `BFTask`. SDK for iOS provides native support for `BFTask`, so that you can chain asynchronous requests instead of nesting

them. This helps keep logic clean and code readable. In this case, the `BFTask` creates a test file in a temporary directory.

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

    BFTask *task = [BFTask taskWithResult:nil];
    [[task continueWithBlock:^id(BFTask *task) {
        // Creates a test file in the temporary directory
        self.testFileURL = [NSURL fileURLWithPath:[NSTemporaryDirectory()
stringByAppendingPathComponent:S3KeyName]];

        NSMutableString *dataString = [NSMutableString new];
        for (int32_t i = 1; i < 2000000; i++) {
            [dataString appendFormat:@"%d\n", i];
        }

        NSError *error = nil;
        [dataString writeToURL:self.testFileURL
                    atomically:YES
                      encoding:NSUTF8StringEncoding
                         error:&error];

        return nil;
    }] continueWithExecutor:[BFExecutor mainThreadExecutor] withBlock:^id(BFTask
 *task) {
        self.uploadButton.enabled = YES;
        self.pauseButton.enabled = YES;
        self.resumeButton.enabled = YES;
        self.cancelButton.enabled = YES;

        return nil;
    }];
```

Having created the task, we need to handle user interactions with the upload, cancel, pause, and resume buttons. First, we define an `uploadButtonPressed` method, where we instantiate `AWSS3TransferManager`, build a request, and pass the request to the `upload:` method.

```
- (IBAction)uploadButtonPressed:(id)sender {
    AWSS3TransferManager *transferManager = [AWSS3TransferManager defaultS3Trans
ferManager];

    self.uploadRequest = [AWSS3TransferManagerUploadRequest new];
    self.uploadRequest.bucket = S3BucketName;
    self.uploadRequest.key = S3KeyName;
    self.uploadRequest.body = self.testFileURL;

    self.uploadStatusLabel.text = StatusLabelUploading;

    [[transferManager upload:self.uploadRequest] continueWithExecutor:[BFExecutor
 mainThreadExecutor] withBlock:^id(BFTask *task) {
        if (task.error != nil) {
            NSLog(@"Error: [%@]", task.error);
            self.uploadStatusLabel.text = StatusLabelFailed;
```

```
        } else {
            self.uploadRequest = nil;
            self.uploadStatusLabel.text = StatusLabelCompleted;
        }
        return nil;
    }];
}
```

Note that `upload:` is an asynchronous method and returns immediately. Since it doesn't block the running thread, it's safe to call this method on the main thread.

We handle the cancel button with a `cancelButtonPressed:` method.

```
- (IBAction)cancelButtonPressed:(id)sender {
   [[self.uploadRequest cancel] continueWithExecutor:[BFExecutor mainThreadEx
ecutor] withBlock:^id(BFTask *task) {
        self.uploadRequest = nil;
        self.uploadStatusLabel.text = StatusLabelReady;
        return nil;
    }];
}
```

We handle the pause button similarly and also check for errors in the upload process:

```
- (IBAction)pauseButtonPressed:(id)sender {
    [[self.uploadRequest pause] continueWithExecutor:[BFExecutor mainThreadEx
ecutor] withBlock:^id(BFTask *task) {
        if(task.error != nil){
            NSLog(@"Error: [%@]", task.error);
            self.uploadStatusLabel.text = StatusLabelFailed;
        } else {
            self.uploadStatusLabel.text = StatusLabelReady;
        }
        return nil;
    }];
}
```

The `resumeButtonPressed:` method is identical to the original `uploadButtonPressed:` method except that we don't need to build the request, since we've already done that:

```
- (IBAction)resumeButtonPressed:(id)sender {
   AWSS3TransferManager *transferManager = [AWSS3TransferManager defaultS3Trans
ferManager];

    self.uploadStatusLabel.text = StatusLabelUploading;

   [[transferManager upload:self.uploadRequest] continueWithExecutor:[BFExecutor
 mainThreadExecutor] withBlock:^id(BFTask *task) {
        if (task.error != nil){
```

```
            NSLog(@"Error: [%@]", task.error);
            self.uploadStatusLabel.text = StatusLabelFailed;
        } else {
            self.uploadRequest = nil;
            self.uploadStatusLabel.text = StatusLabelCompleted;
        }
        return nil;
    }];
}
```

Of course, there's much more that you can do with Amazon S3. If you'd like to download the latest sample code for Amazon S3 `TransferManager` and other supported AWS services, go to the GitHub repository for AWS SDK for iOS.

How to Store Objects in the Cloud

Using the DynamoDB Object Mapper, you can store application data to an Amazon DynamoDB database in the cloud. The DynamoDB Object Mapper makes it easy to connect to the database, and it supports high-level operations like creating, getting, querying, updating, and deleting records.

What is 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.

The AWS Mobile SDKs provide a high-level library for working with DynamoDB. You can also make requests directly against the low-level DynamoDB API, but for most use cases the high-level library is recommended.

The DynamoDB Object Mapper is an especially useful part of the high-level library. Using the Object Mapper, you can map client-side classes to DynamoDB tables; perform various create, read, update, and delete (CRUD) operations; and execute queries.

Integrating Amazon DynamoDB

To use DynamoDB in an iOS application, you'll need to integrate the SDK for iOS into your app and import the necessary libraries. The following steps walk you through the process of integrating the SDK with an existing app.

> **Note**
> If you'd rather try out DynamoDB in an example app, you can download the DynamoDB sample app from the iOS Samples repo.

1. If you haven't already done so, download the SDK for iOS, unzip it, and include it in your application as described at Include the SDK for iOS in an Existing Application (p. 3). The instructions direct you to import the headers for the services you'll be using. For this example, you need the following import:

```
#import <AWSiOSSDKv2/DynamoDB.h>
```

2. 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 (p. 13).
3. To use DynamoDB in an application, you must set the correct permissions. The following IAM policy allows the user to delete, get, put, scan, and update items in a specific DynamoDB table, which is identified by ARN:

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

This policy should be applied to roles assigned to the Cognito identity pool, but you will need to replace the `Resource` value with the correct ARN for your DynamoDB table. Cognito automatically creates a role for your new identity pool, and you can apply policies to this role at the IAM console. You should add or remove allowed actions based on the needs of your app. To learn more about IAM policies, see Using IAM. To learn more about DynamoDB-specific policies, see Using IAM to Control Access to DynamoDB Resources.

Create a DynamoDB Table

Now that we have our permissions and credentials set up, let's create a DynamoDB table for our application. We can create the table in our code using the `createTable:` method, or through the console. Let's use the console. To create a DynamoDB table, go to the DynamoDB console and follow these steps:

1. Click **Create Table**.
2. Enter **`Bookstore`** as the name of the table.
3. Select **Hash** as the primary key type.
4. Select **String** and enter **`bookId`** for the hash attribute name. Click **Continue**.
5. Click **Continue** again to skip adding indexes.
6. Set the read capacity to **`10`** and the write capacity to **`5`**. Click **Continue**.
7. Enter a notification email and click **Continue** to create throughput alarms.
8. Click **Create**. DynamoDB will create your database.

Create a DynamoDB Client

For our app to interact with a DynamoDB table, we need a client. We can create a default DynamodDB client as follows:

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

The `AWSDynamoDB` class is the entry point for the DynamoDB API. The class provides instance methods for creating, describing, updating, and deleting tables, among other operations.

Describe a Table

The SDK for iOS provides native support for `BFTask`, a Bolts class that makes it easier to manage asynchronous operations and background threads. To get a description of our DynamoDB table, we can use the following code block, which employs `BFTask`:

```
AWSDynamoDB *dynamoDB = [AWSDynamoDB defaultDynamoDB];
AWSDynamoDBDescribeTableInput *describeTableInput = [AWSDynamoDBDescribeTableIn
put new];
describeTableInput.tableName = @"Bookstore";
[[dynamoDB describeTable:describeTableInput] continueWithSuccessBlock:^id(BFTask
 *task) {
    AWSDynamoDBDescribeTableOutput *describeTableOutput = task.result;
    NSString *tableDescription = describeTableOutput.table.description;
    //Do something with tableDescription...
    return nil;
}];
```

In this example, we create a client and an `AWSDynamoDBDescribeTableInput` object, assign the name of our table to the `tableName` property of the input object, and then pass the input object to the `describeTable:` method. By putting the rest of the table description logic in `continueWithSuccessBlock:`, we ensure that the result of `describeTable:` will have returned before we assign it to `AWSDynamoDBDescribeTableOutput`. Note that the return value of `describeTable:` is an instance of `BFTask`. When the method executes successfully, `task.result` will contain an instance of `AWSDynamoDBDescribeTableOutput`.

All of the `AWSDynamoDB` methods return instances of `BFTask`. Thus, we can use the `describeTable:` example above as a model for other method calls.

To learn more about working with BFTask, see The AWS Mobile SDK for iOS - How to Use BFTask.

Create a DynamoDB Object Mapper

The SDK for iOS provides an Object Mapper class, `AWSDynamoDBObjectMapper`, that supports high-level operations like creating, getting, querying, updating, and deleting records. We can create an Object Mapper as follows:

```
AWSDynamoDBObjectMapper *dynamoDBObjectMapper = [AWSDynamoDBObjectMapper default
DynamoDBObjectMapper];
```

The Object Mapper provides a number of instance methods, including `save:`, `remove:`, `load:hashKey:rangeKey:`, `query:expression:`, and `scan:expression:`. Like the `AWSDynamoDB` methods, all of the Object Mapper methods return a `BFTask` object.

Save an Object

The `save:` method saves an object to DynamoDB, using the default configuration. `save:` takes as a parameter an object that inherits from `AWSDynamoDBModel` and conforms to the `AWSDynamoDBModeling` protocol. The properties of this object will be mapped to attributes in the DynamoDB table. Let's see what this looks like.

We called our database "Bookstore", and in keeping with that theme we'll implement a data model that records book-related attributes. Here's `XYZDatabaseModel.h`, the header for our data model class:

```
#import <Foundation/Foundation.h>
#import <AWSiOSSDKv2/DynamoDB.h>

@interface DDBTableRow : AWSDynamoDBModel <AWSDynamoDBModeling>

@property (nonatomic, assign) NSString *bookId;
@property (nonatomic, strong) NSString *isbn;
@property (nonatomic, strong) NSString *title;

@end
```

(Of course, for a real bookstore application we'd need additional fields for things like author and price.)

And here's the implementation of our model:

```
#import <AWSiOSSDKv2/DynamoDB.h>
#import "XYZDatabaseModel.h"

@implementation DDBTableRow

+ (NSString *)dynamoDBTableName {
    return @"Bookstore";
}

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

+ (NSString *)rangeKeyAttribute {
    return @"rangeKey";
}

@end
```

To conform to `AWSDynamoDBModeling`, we have to implement `dynamoDBTableName` and `hashKeyAttribute`.

With the model in place, we can construct a `DDBTableRow` object and pass it to the `save:` method:

```
```

```
DDBTableRow *tableRow = [DDBTableRow new];
tableRow.bookId = @"31";
tableRow.isbn = @"222-222222-222";
tableRow.title = @"Some Book Title";

AWSDynamoDBObjectMapper *dynamoDBObjectMapper = [AWSDynamoDBObjectMapper default
DynamoDBObjectMapper];
[[dynamoDBObjectMapper save:tableRow]
continueWithExecutor:[BFExecutor mainThreadExecutor]
          withBlock:^id(BFTask *task) {
   //This block will be executed after the table row is saved.
   return nil;
}];
```

To update a table row, we would also use the `save:` method. We would just pass in a `DDBTableRow` object with the `bookId` (that is, the hash key) of the item to be updated and with updated attribute values.

Delete a Table Row

To delete a table row, use the `remove:` method:

```
[[dynamoDBObjectMapper remove:tableRow]
```

How to Stream Data for Real-Time Processing

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

The SDK for iOS provides a high-level client, `AWSKinesisRecorder`, designed to help you interface with Amazon Kinesis. The Kinesis client lets you store `PutRecord` requests on disk and then send them all at once. This is useful because many mobile applications that use 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 Kinesis client for batching can preserve both battery life and data.

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.

Integrating Amazon Kinesis

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:

1. If you haven't already done so, download the SDK for iOS, unzip it, and include it in your application as described at Include the SDK for iOS in an Existing Application (p. 3). The instructions direct you to import the headers for the services you'll be using. For this example, you need the following import:

```
#import <AWSiOSSDKv2/Kinesis.h>
```

2. 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 (p. 13).
3. 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 Kinesis stream, which is identified by ARN:

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

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

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

```
AWSKinesisRecorder *kinesisRecorder = [AWSKinesisRecorder defaultKinesisRecord
er];
```

You can configure `AWSKinesisRecorder` through its properties:

```
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 Kinesis doesn't post the notification.

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

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

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

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.

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

```
[kinesisRecorder submitAllRecords];
```

`submitAllRecords` sends all locally saved requests to the 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:

```
// 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 stringWith
Format:@"TestString-%02d", i] dataUsingEncoding:NSUTF8StringEncoding]
                                      streamName:@"YourStreamName"]];
}

[[[BFTask taskForCompletionOfAllTasks:tasks] continueWithSuccessBlock:^id(BFTask
 *task) {
    return [kinesisRecorder submitAllRecords];
}] continueWithBlock:^id(BFTask *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 Kinesis classes, see the class reference for AWSKinesisRecorder.

Document History

The following table describes the important changes since the last release of the *AWS SDK for iOS Getting Started Guide*.

Last documentation update: June 30, 2014

Change	Description	Release Date
SDK for iOS v2	The *AWS SDK for iOS Getting Started Guide* has been updated to document SDK for iOS v2.	June 30, 2014
New topic	This topic tracks recent changes to the *AWS SDK for iOS Getting Started Guide*. It is intended as a companion to the release notes history.	September 9, 2013