AWS SDK for Android 2.0

Developer Guide

AWS SDK for Android 2.0: Developer Guide

Table of Contents

What Is the AWS SDK for Android?

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

Note
This guide describes AWS SDK for Android 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 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 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. 7) and How to Sync Profile Data (p. 13).

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. 17).

About the SDK for Android

The AWS SDK for Android includes the following:

- **Class libraries** – Classes 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 class libraries for only the services you need and keep your application as small as possible.
- **Code samples** – Practical examples of using the class libraries to build applications.
- **Documentation** – Reference documentation for the AWS SDK for Android.

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

- **License.txt**
- **Notice.txt**
- **Readme.txt**
- **lib/**
  - **release**–Contains Java archive files (`.jar`) that include AWS class libraries. To manage the size of your application, you can include only the files that you need for the services your application is using. Note that the *`2.0.0`* number, indicated in *`red`*, can change.
    - **aws-android-sdk-*2.0.0*-core.jar**–Contains all third-party dependencies and the base platform used to send and process requests to AWS services. This is required for any use of the SDK.
    - **aws-android-sdk-*2.0.0*-core-no-third-party.jar**–Contains the base platform used to send to and process requests from AWS services but does not contain any of the third-party dependencies.
    - **aws-android-sdk-*2.0.0*-mobileanalytics.jar**–Contains the classes necessary to use Amazon Mobile Analytics. Must be used with the `aws-android-sdk-2.0.0-core.jar`.
    - **aws-android-sdk-*2.0.0*-kinesis.jar**–Contains the classes necessary to use Amazon Kinesis. Must be used with `aws-android-sdk-2.0.0-core.jar`..
    - **aws-android-sdk-*2.0.0*-s3.jar**–Contains the classes necessary to use Amazon Simple Storage Service (Amazon S3). Must be used with the `aws-android-sdk-2.0.0-core.jar` file.
    - **aws-android-sdk-*2.0.0*-sqs.jar**–Contains the classes necessary to use Amazon Simple Queue Service (Amazon SQS). Must be used with the `aws-android-sdk-2.0.0-core.jar` file.
    - **aws-android-sdk-*2.0.0*-sns.jar**–Contains the classes necessary to use Amazon Simple Notification Service (Amazon SNS). Must be used with the `aws-android-sdk-2.0.0-core.jar` file.
    - **aws-android-sdk-*2.0.0*-ses.jar**–Contains the classes necessary to use Amazon Simple Email Service (Amazon SES). Must be used with the `aws-android-sdk-2.0.0-core.jar` file.
    - **aws-android-sdk-*2.0.0*-sdb.jar**–Contains the classes necessary to use Amazon SimpleDB. Must be used with the `aws-android-sdk-2.0.0-core.jar` file.
    - **aws-android-sdk-*2.0.0*-ddb-mapper.jar**–Contains all the classes necessary to use the Amazon DynamoDB object mapper. Must be used with `aws-android-sdk-2.0.0-core.jar` and `aws-android-sdk-2.0.0-ddb.jar` files.
    - **aws-android-sdk-*2.0.0*-ddb.jar**–Contains all the classes necessary to use Amazon DynamoDB. Must be used with `aws-android-sdk-2.0.0-core.jar` file.
    - **aws-android-sdk-*2.0.0*-autoscaling.jar**–Contains the classes necessary to use Auto Scaling. Must be used with the `aws-android-sdk-2.0.0-core.jar` file.
    - **aws-android-sdk-*2.0.0*-cloudwatch.jar**–Contains the classes necessary to use Amazon CloudWatch. Must be used with the `aws-android-sdk-2.0.0-core.jar` file.

  - **aws-android-sdk-*2.0.0*-ec2.jar**– Contains the classes necessary to use Amazon Elastic Compute Cloud (Amazon EC2). Must be used with the `aws-android-sdk-2.0.0-core.jar` file.
  - **aws-android-sdk-*2.0.0*-elb.jar**– Contains the classes necessary to use Elastic Load Balancing. Must be used with the `aws-android-sdk-2.0.0-core.jar` file.
  - **extras/**
    - **aws-android-sdk-*2.0.0*-cognito.jar**–Contains classes necessary to use Amazon Cognito.
    - **Amazon Software License.txt**
    - **NOTICE - CognitoSync.txt**
- **debug/**–Contains Java archive files (`.jar`) that include AWS class libraries. To manage the size of your application, you can include only the files that you need for the services your application is using. The code in the `debug` directory was compiled with the debug option on with all symbols retained. You can use the debug versions of JARs in place of the release versions listed above. The debug versions help with debugging, as you will be able to see the line numbers of code breaks, but the release JARs are smaller files. Note that the *2.0.0* number, indicated in *red*, can change.
  - **aws-android-sdk-*2.0.0*-core.debug.jar**–Contains all third-party dependencies and the base platform used to send and process requests to AWS services. This is required for any use of the SDK.
  - **aws-android-sdk-*2.0.0*-core-no-third-party.debug.jar**–Contains the base platform used to send to and process requests from AWS services but does not contain any of the third-party dependencies.
  - **aws-android-sdk-*2.0.0*-mobileanalytics.debug.jar**–Contains the classes necessary to use Amazon Mobile Analytics. Must be used with the `aws-android-sdk-2.0.0-core.jar`.
  - **aws-android-sdk-*2.0.0*-kinesis.debug.jar**–Contains the classes necessary to use Amazon Kinesis. Must be used with `aws-android-sdk-2.0.0-core.jar`..
  - **aws-android-sdk-*2.0.0*-s3.debug.jar**–Contains the classes necessary to use Amazon Simple Storage Service (Amazon S3). Must be used with the `aws-android-sdk-2.0.0-core.jar`
  - **aws-android-sdk-*2.0.0*-sqs.debug.jar**–Contains the classes necessary to use Amazon Simple Queue Service (Amazon SQS). Must be used with the `aws-android-sdk-2.0.0-core.jar` file.
  - **aws-android-sdk-*2.0.0*-sns.debug.jar**–Contains the classes necessary to use Amazon Simple Notification Service (Amazon SNS). Must be used with the `aws-android-sdk-2.0.0-core.jar` file.
  - **aws-android-sdk-*2.0.0*-ses.debug.jar**–Contains the classes necessary to use Amazon Simple Email Service (Amazon SES). Must be used with the `aws-android-sdk-2.0.0-core.jar` file.
  - **aws-android-sdk-*2.0.0*-sdb.debug.jar**–Contains the classes necessary to use Amazon SimpleDB. Must be used with the `aws-android-sdk-2.0.0-core.jar` file.
  - **aws-android-sdk-*2.0.0*-ddb-mapper.debug.jar**–Contains all the classes necessary to use the Amazon DynamoDB object mapper. Must be used with `aws-android-sdk-2.0.0-core.jar` and `aws-android-sdk-2.0.0-ddb.jar` files.
  - **aws-android-sdk-*2.0.0*-ddb.debug.jar**–Contains all the classes necessary to use Amazon DynamoDB. Must be used with `aws-android-sdk-2.0.0-core.jar` file.
  - **aws-android-sdk-*2.0.0*-autoscaling.debug.jar**–Contains the classes necessary to use Auto Scaling. Must be used with the `aws-android-sdk-2.0.0-core.jar` file.
  - **aws-android-sdk-*2.0.0*-cloudwatch.debug.jar**–Contains the classes necessary to use Amazon CloudWatch. Must be used with the `aws-android-sdk-2.0.0-core.jar` file.
  - **aws-android-sdk-*2.0.0*-ec2.debug.jar**– Contains the classes necessary to use Amazon Elastic Compute Cloud (Amazon EC2). Must be used with the `aws-android-sdk-2.0.0-core.jar` file.
  - **aws-android-sdk-*2.0.0*-elb.debug.jar**– Contains the classes necessary to use Elastic Load Balancing. Must be used with the `aws-android-sdk-2.0.0-core.jar` file.
  - **extras/**

  - **aws-android-sdk-*2.0.0*-cognito.debug.jar**–Contains classes necessary to use Amazon Cognito.
  - **Amazon Software License.txt**
  - **NOTICE - CognitoSync.txt**
- **documentation/**–Includes Javadoc files and other documentation for using the AWS SDK for Android.
- **samples/**–Contains an HTML document with links to samples on GitHub. Samples are named based on the services they demonstrate.
- **src/**–Contains an HTML document with links to source on GitHub. Contains the original source files for the class libraries.
- **third-party/**–Contains third-party libraries.

Important
The SDK for Android no longer includes a separate JAR for AWS Security Token Service. AWS STS is now bundled with the core, and including STS as a separate JAR will result in a compile-time error.

Next Steps

- To start using the SDK for Android, follow the setup instructions at Set Up the SDK for Android (p. 5).
- For more information about the AWS SDK for Android, 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 across all code included with the SDK. It provides thorough documentation and usage examples. You can find it at AWS SDK for Android API Reference.
- Post questions and feedback at the Mobile Developer Forum.
- Source code and sample applications are available at the AWS SDK for Android project on GitHub. Source code for Amazon Cognito is available at https://github.com/aws/amazon-cognito-android.

Set Up the SDK for Android

To get started with the AWS SDK for Android, 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 Android and set up a credential provider using Amazon Cognito.

Get the SDK for Android

In order to use the AWS SDK for Android, you will need the following:

- Android 2.3 (API Level 10) or higher. For more information on the SDK for Android, see http://developer.android.com/index.html.
- Android Development Tools for running code examples.

Download the SDK from http://aws.amazon.com/mobile/sdk. The SDK is stored in a compressed file named `aws-android-sdk-#-#-#.zip`, where #-#-# represents the version number.

Source code is available on GitHub.

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. 7).

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.

Include the SDK for Android in an Existing Application

To use AWS with an existing application, follow these steps:

1. Create a `libs` directory within your Android project directory, if one does not already exist.
2. Drag `aws-android-sdk-VERSION-core.jar`, plus the `.jar` files for the individual services your project will use, into the `libs` folder. They'll be included on the build path automatically.
3. Ensure that the following permission is set in `AndroidManifest.xml`:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Run the Demos

At the AWSLabs repo, you can find sample Android apps that demonstrate common use cases.

To run the sample apps, set up the SDK for Android as described above, and then follow the instructions contained in the README files of the individual samples.

Getting Started with Services

You can use the AWS SDK for Android to build Android 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.

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. 13)

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
- AWS Mobile SDK for Android installed on your development machine.
- Optionally, a developer account and an application registered with the identity provider you want to use (Facebook, Google, or Amazon).
- The following `.jar` files included in your project (where *`red text`* is replaceable by the current version number):
  - `aws-android-sdk-2.0.0-cognito.jar`
  - `aws-android-sdk-2.0.0-core.jar`
- Or, to debug, include the following files instead:
  - `aws-android-sdk-2.0.0-cognito.debug.jar`
  - `aws-android-sdk-2.0.0-core.debug.jar`

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 `setLogins` or `withLogins` methods to set up credentials received from an identity provider. The instructions below guide you through authentication with the identity providers supported by Amazon Cognito.

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 Android 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 user with the Facebook SDK, add the session token to the Amazon Cognito credentials provider.

```
Map<String, String> logins = new HashMap<String, String>();
logins.put("graph.facebook.com",Session.getActiveSession().getAccessToken());
provider.withLogins(logins);
```

The Facebook login process initializes a singleton session within its SDK. The Facebook session object contains an OAuth 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 Android. A successful authentication with Google results in an Open ID authentication token. The Amazon Cognito SDK uses this token to authenticate the user and generate the unique identifier.

The following sample code shows how to retrieve the authentication token from the Google Play Service.

```
GooglePlayServicesUtil.isGooglePlayServicesAvailable(getApplicationContext());
AccountManager am = AccountManager.get(this);
Account[] accounts = am.getAccountsByType(GoogleAuthUtil.GOOGLE_ACCOUNT_TYPE);
String token = GoogleAuthUtil.getToken(getApplicationContext(), accounts[0].name,

    "audience:server:client_id:YOUR_GOOGLE_CLIENT_ID");
Map<String, String> logins = new HashMap<String, String>();
logins.put("accounts.google.com", token);
provider.withLogins(logins);
```

Amazon

The Login with Amazon guide takes you through the process of setting up Login with Amazon in your application, downloading the client SDK, and declaring your application on the Amazon developer platform.

Once login with Amazon is implemented, you can pass the token to the Amazon Cognito credentials provider in the `onSuccess` method of the `TokenListener` interface. The code looks like this:

```
@Override
public void onSuccess(Bundle response) {
    String token = response.getString(AuthzConstants.BUNDLE_KEY.TOKEN.val);
    Map<String, String> logins = new HashMap<String, String>();
    logins.put("www.amazon.com", token);

    provider.withLogins(logins);
}
```

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 Amazon Cognito 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 `aws-android-sdk-2.X.X-core.jar` to your project. (For more information, see Include the SDK for Android in an Existing Application (p. 6)).
3. Include the following import statements:

```
import com.amazonaws.auth.CognitoCachingCredentialsProvider;
import com.amazonaws.regions.Regions;
```

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:

```
CognitoCachingCredentialsProvider cognitoProvider = new CognitoCachingCre
dentialsProvider(
    myActivity.getContext(), // get the context for the current activity
    "XXXXXXXXXXXX",
    "us-east-1:XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXX",
    "arn:aws:iam::XXXXXXXXXX:role/YourUnauthRoleName",
    "arn:aws:iam::XXXXXXXXXX:role/YourAuthRoleName",
    Regions.US_EAST_1
);
```

5. Pass the initialized Amazon Cognito credentials provider to the initializer for an AWS client. The code required depends on the service to be initialized.

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.

The unique identifier is available in the `identityId` property of the credentials provider object. You can retrieve it using a get method, as in this code:

```
Log.d("LogTag", "my ID is " + provider.getIdentityId());
```

Once the Cognito credentials provider is initialized and refreshed, you can pass it directly to the initializer for an AWS client:

```
// Create a service client with the provider
AmazonDynamoDB client = new AmazonDynamoDBClient(provider);
```

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. The provider refreshes them when they expire.

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 ensures that the authenticated identity retains the same unique identifier as the new, authenticated one, and that profile data is merged automatically.

Your application is informed of a profile merge through the `IdentityChangedListener` interface. Implement the `identityChanged` method in the interface to receive these messages.

```
@override
public void identityChanged(String oldIdentityId, String newIdentityId) {
    // handle the change
}
```

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.

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. 12).

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. You need to import `com.amazonaws.mobileconnectors.cognito.*` and provide an initialized Amazon Cognito credentials provider object and the Android Activity context. Amazon Cognito reads the user information from the credentials provider you pass to the constructor.

```
import com.amazonaws.mobileconnectors.cognito.*;
CognitoSyncManager syncClient = new CognitoSyncManager(myActivity.getContext(),
 COGNITO_POOL_ID, Regions.YOUR_REGION, provider);
```

Datasets

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

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

To create a new dataset or open an existing one, use the following code:

```
Dataset dataset = client.openOrCreateDataset("datasetname");
```

To delete a dataset, call the delete method:

```
dataset.delete();
```

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.

```
String value = dataset.get("myKey");

dataset.put("myKey", "my value");
```

You can also use the `remove` method to remove keys from a data set.

```
dataset.remove("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.

```
dataset.synchronize(this, syncCallback);
```

The synchronize method receives an implementation of the `SyncCallback` interface.

The `synchronizeOnConnectivity()` method attempts to synchronize when connectivity is available. If connectivity is immediately available, `synchronizeOnConnectivity()` behaves like `synchronize()`. Otherwise it listens for connectivity changes and performs a sync once connectivity is available. Note that if this method is called multiple times, only the last synchronize request is kept, and only the last callback will fire. If either the dataset or the callback is garbage collected, this method will not perform a sync, and the callback won't fire.

SyncCallback Interface

Applications can receive notifications from Amazon Cognito datasets by implementing the `SyncCallback` interface. Through the callback, applications can make active decisions regarding the deletion of local data, the merging of unauthenticated and authenticated profiles, and the resolution of synchronization conflicts.

The following methods are required in the interface and should be implemented:

- `onSuccess`
- `onFailure`
- `onConflict`
- `onDatasetDeleted`
- `onDatasetsMerged`

The success callback is triggered when a dataset is successfully downloaded from the cloud sync store.

```
@override
public void onSuccess(Dataset dataset, List<Record> newRecords) {
}
```

The failure method is called if an exception occurs during the synchronization operation.

```
@override
public void onFailure(DataStorageException dse) {
}
```

Conflicts may arise during the sync operation if the same key on the local store and cloud sync store have been modified. The `onConflict` method handles conflict resolution. If the listener method is not implemented, the Cognito Sync client defaults to using the most recent change.

```
@Override
public boolean onConflict(Dataset dataset, final List<SyncConflict> conflicts)
 {
    List<Record> resolvedRecords = new ArrayList<Record>();
    for (SyncConflict conflict : conflicts) {
        /* resolved by taking remote records */
        resolvedRecords.add(conflict.resolveWithRemoteRecord());

        /* alternately take the local records */
        // resolvedRecords.add(conflict.resolveWithLocalRecord());

        /* or customer logic, say concatenate strings */
        // String newValue = conflict.getRemoteRecord().getValue()
        //     + conflict.getLocalRecord().getValue();
        // resolvedRecords.add(conflict.resolveWithValue(newValue);
    }
    dataset.resolve(resolvedRecords);

   // return true so that synchronize() is retried after conflicts are resolved

    return true;
}
```

The Amazon Cognito client uses the callback interface to confirm whether the local cached copy of the dataset should be deleted too.

Implement the `onDatasetDeleted` method from the interface to tell the client SDK what to do with the local data.

```
@override
public boolean onDatasetDeleted(Dataset dataset, String datasetName) {
    // return true to delete the local copy of the dataset
    return true;
}
```

When two previously unconnected identities are linked together, all of their Datasets are merged.

Applications are notified of the merge through the `onDatasetsMerged` method of the callback interface.

```
@override
public boolean onDatasetsMerged(Dataset dataset, List<String> datasetNames) {
   // return false to handle Dataset merge outside the synchronization callback

    return false;
}
```

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 Android 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

The sections below explain how to integrate Mobile Analytics with your app, without requiring users of your app to be authenticated with Google, Facebook, or Amazon.

Create an Amazon Cognito Identity Pool for Unauthenticated Users

At the Amazon Cognito console, create a new identity pool.

1. Provide a unique name (for example, your app name) for the identity pool.
2. Leave the fields for identity providers blank.
3. Check the box for **Enable Access to Unauthenticated Identities**.
4. Click **Create Pool**.
5. Create new IAM roles for Authenticated and Unauthenticated identities. The default configuration in the Cognito console is correct for this example, so you can just click **Update Roles**.
6. Click **Download Starter Code (Android and iOS)** to download the code necessary to integrate Cognito with your app. In the next section, you'll need the Cognito initialization snippet for reference.

Integrate the SDK into Your App

1. Download the SDK and unzip it.
2. Add the following jars from `aws-android-sdk-2.x.x\lib\release` to a `libs` folder in your Android project:
   - `aws-android-sdk-x.x.x-core.jar`
   - `aws-android-sdk-x.x.x-mobileanalytics.jar`
3. Make sure that the new libraries are included when building your app. (The specific procedure depends on the IDE you're using.)
4. In `AndroidManifest.xml`, set the following permissions, if they're not aready present:

```
<uses-permission android:name="android.permission.INTERNET" />
```

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

5. Add the following imports to the main activity of your app.

   ```
   import com.amazonaws.mobileconnectors.amazonmobileanalytics.*;
   import com.amazonaws.auth.CognitoCachingCredentialsProvider;
   import com.amazonaws.regions.Regions;
   ```

   For this example, we're going to use the Log class, so import that, too:

   ```
   import android.util.Log;
   ```

6. Define a static variable to hold a reference to the Mobile Analytics client.

   ```
   private static MobileAnalyticsManager analytics;
   ```

   For this particular example, let's also create two constants that we'll use later in a custom event:

   ```
   private static final int STATE_LOSE = 0;
   private static final int STATE_WIN = 1;
   ```

7. In the activity's `onCreate()` method, initialize the Cognito client and then create an instance of `MobileAnalyticsManager`. You'll need to replace `AWS_ACCOUNT_ID`, `COGNITO_IDENTITY_POOL`, `UNAUTHENTICATED_ROLE` and `AUTHENTICATED_ROLE` with applicable values from the Cognito snippet you downloaded previously.

   ```
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       // Create a credentials provider.
       // Replace AWS_ACCOUNT_ID, COGNITO_IDENTITY_POOL, UNAUTHENTICATED_ROLE
    and AUTHENTICATED_ROLE with the applicable values.
      CognitoCachingCredentialsProvider cognitoProvider = new CognitoCachingCre
   dentialsProvider(
           getApplicationContext(),
           AWS_ACCOUNT_ID,
           COGNITO_IDENTITY_POOL,
           "arn:aws:iam::AWS_ACCOUNT_ID:role/UNAUTHENTICATED_ROLE",
           "arn:aws:iam::AWS_ACCOUNT_ID:role/AUTHENTICATED_ROLE",
           Regions.US_EAST_1
       );

       try {
           AnalyticsConfig = new AnalyticsConfig();
   ```

```
        options.withAllowsWANDelivery(true);
        analytics = MobileAnalyticsManager.getOrCreateInstance(
                This.getApplicationContext(),
                "yourCompany.yourAppId", Regions.YOUR_REGION, config, cog
nitoProvider );
    } catch(InitializationException ex) {
            Log.e(this.getClass().getName(), "Failed to initialize Amazon
Mobile Analytics", ex);
    }
}
```

By default, the `MobileAnalyticsManager` client initializes with WAN delivery disabled. But to run Mobile Analytics on an Android emulator, you must have WAN delivery enabled, because the emulator only recognizes internet connectivity as WAN. You can enable WAN in the client by passing in a correctly configured `AnalyticsOptions` object, as shown above.

The second parameter of the `MobileAnalyticsManager().getOrCreateInstance()` method (in the example above, "yourCompany.yourAppId") should be the unique namespace of your app. The namespace is used in several ways, including as the name of a directory where `MobileAnalyticsManager` stores event-related files.

8. Override the activity's `onPause()` and `onResume()` methods to record session events.

```

/**
 * Invoked when the Activity loses user focus.
 */
@Override
protected void onPause() {

    super.onPause();
    if(analytics != null) {
        analytics.getSessionClient().pauseSession();
          //Attempt to send any events that have been recorded to the Mobile
 Analytics service.
        analytics.getEventClient().submitEvents();
    }
}

@Override
protected void onResume() {
    super.onPause();
    if(analytics != null)  {
        analytics.getSessionClient().resumeSession();
    }
}
```

For each activity in your application, you will need to record session events in the onPause() and onResume() methods.

9. The Mobile Analytics client lets us create and record custom events. For example, if our app were a game, we might create a custom event to be submitted when the user completes a level. In your main activity, add the following method, which creates and records a custom event.

```
/**
 * This method gets called when the player completes a level.
 * @param levelName the name of the level
 * @param difficulty the difficulty setting
 * @param timeToComplete the time to complete the level in seconds
 * @param playerState the winning/losing state of the player
 */
public void onLevelComplete(String levelName, String difficulty, double
timeToComplete, int playerState) {

    //Create a Level Complete event with some attributes and metrics(meas
urements)
    //Attributes and metrics can be added using with statements
    AnalyticsEvent levelCompleteEvent = analytics.getEventClient().cre
ateEvent("LevelComplete")
            .withAttribute("LevelName", levelName)
            .withAttribute("Difficulty", difficulty)
            .withMetric("TimeToComplete", timeToComplete);

    //attributes and metrics can also be added using add statements
    if (playerState == STATE_LOSE)
        levelCompleteEvent.addAttribute("EndState", "Lose");
    else if (playerState == STATE_WIN)
        levelCompleteEvent.addAttribute("EndState", "Win");

    //Record the Level Complete event
    analytics.getEventClient().recordEvent(levelCompleteEvent);
}
```

10. Test the custom event by calling it at the end of the `onCreate()` method:

```
this.onLevelComplete("Lower Dungeon", "Very Difficult", 2734, STATE_WIN);
```

11. Launch and test your app.
12. Navigate to the Mobile Analytics console. Your app should appear in the drop-down list, though it may take a few minutes for a new app to appear in the list. You should see session-related data for your app in the graphs.

How to Upload Files from Your App to the Cloud

The Amazon Simple Storage Service `TransferManager` simplifies the process of transferring files to and from Amazon S3. Using the Amazon S3 `TransferManager`, you can transfer files from your app on a 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. Transfer-Manager 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.

Create a Client

To use S3 `TransferManager`, you'll need to have an instance of `AWSCredentialsProvider`. For this example, we'll use Amazon Cognito as a credential provider.

```
public static AWSCredentialsProvider getCredProvider(Context appContext) {
    if(sCredProvider == null) {
        sCredProvider = new CognitoCachingCredentialsProvider(
                appContext,
                Constants.AWS_ACCOUNT_ID,
                Constants.COGNITO_POOL_ID,
                Constants.COGNITO_ROLE_UNAUTH,
                null,
                Regions.US_EAST_1);
        sCredProvider.refresh();
    }
    return sCredProvider;
}
```

Note that the last parameter takes an AWS region, so our sample app should include the following import statement, among others:

```
import com.amazonaws.regions.Regions;
```

Now that we have the credentials, we can make an instance of the `TransferManager`.

```
TransferManager transferManager = new TransferManager(credProvider);
```

Download from S3

To download a file, you need to know the bucket name and key name of the file. So let’s get the keys first:

```
AmazonS3Client mClient = new AmazonS3Client(credProvider);
List<S3ObjectSummary> summaries = mClient.listObjects(BUCKET_NAME).getObjectSum
maries();
String[] keys = new String[summaries.size()];
for(int i = 0; i < keys.length; i++) {
    keys[i] = summaries.get(i).getKey();
}
```

Here we get the `S3ObjectSummary` objects first, and then we get the keys from them. The `BUCKET_NAME` is the name of the bucket you want to get keys from. Now that we have the keys, we can do a download with a single statement:

```
Download download = transferManager.download(BUCKET_NAME, mKey, file);
```

Here, `mKey` is the key of the file to download, and `file` is a `java.io.File` to which to write the file. Note that we are given a `Download` object, which we can use to pause or abort the download, among other things.

Upload to S3

Uploading is very similar to downloading, and you can do it as follows:

```
Upload upload = transferManager.upload(
        BUCKET_NAME,
        fileName,
        file);
```

Similar to the downloading case, you are passed an `Upload` object that you can use to pause or abort the upload.

Abort an S3 Transfer

Aborting an upload or download is simple. Just call `abort()` on the upload or download:

```
upload.abort();
download.abort();
```

Pause or Resume an S3 Transfer

There are two different ways to pause: `pause()` and `tryPause(boolean)`. The latter method is necessary because it's not always possible to pause a transfer.

Transferring happens in chunks, so if a file is split into multiple chunks for transfer and we want to pause, we can just throw away our current chunk and start from there. However, if a file is transferred as only one chunk, and we follow the same method, we are just throwing away the whole file, thus making it equivalent to an abort. We can only pause transfers that happen in chunks, which means that we can't pause transfers that are opened with an `InputStream`. Encryption can also prevent pausing.

If you call pause and you aren't aborting, you'll get a `PersistableUpload` or `PersistableDownload`, depending on the transfer type. However, If you call `pause()` but you're just doing an abort, then the transfer will stop and a `PauseException` will be thrown. Note that in the case of an exception being thrown, there will be no return value. On the other hand, `tryPause(boolean)` gives you more control over the outcome of an abort. The parameter determines whether the transfer will abort or just continue, and you can use the return value to see what happened.

Here's an example of how we would use `pause()`.

```
try {
    PersistableUpload persistableUpload = upload.pause();
    //do something if we didn’t abort
} catch(PauseException e) {
    //do something if we aborted
}
```

Now, if we want to resume, we just need to use the `PersistableUpload` or `PersistableDownload` object as follows:

```
Upload resumedUpload = transferManager.resumeUpload(persistableUpload);
```

For both examples we used an `Upload`, but it’s identical in the case of downloads; you would just use `resumeDownload` and would get a `Download` object back instead.

How to Store Objects in the Cloud

Using the DynamoDB Object Mapper, you can write simple, readable code that stores Java objects in Amazon DynamoDB.

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 the DynamoDB Object Mapper, you'll need to integrate the SDK for Android into your app and import the necessary libraries. To do so, follow these steps:

1. Download the SDK and unzip it.
2. Add the following JARs from `aws-android-sdk-2.x.x\lib\release` to a `libs` folder in your Android project:
   - `aws-android-sdk-x.x.x-core.jar`
   - `aws-android-sdk-x.x.x-ddb-mapper.jar`
   - `aws-android-sdk-x.x.x-ddb.jar`
3. Make sure that the new libraries are included when building your app. The specific procedure depends on the IDE you're using. If you're working in Eclipse, you can simply drag the JARs into your `libs` folder, and Eclipse will adjust the build path for you.

4. In `AndroidManifest.xml`, set the following permission, if it's not aready present:

   ```
   <uses-permission android:name="android.permission.INTERNET" />
   ```

5. Add the following imports to the main activity of your app.

   ```
   import com.amazonaws.auth.CognitoCachingCredentialsProvider;
   import com.amazonaws.services.dynamodbv2.*;
   import com.amazonaws.services.dynamodbv2.datamodeling.*;
   import com.amazonaws.services.dynamodbv2.model.*;
   ```

6. 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. 12).
7. To use DynamoDB in an application, you must set the correct permissions. The following IAM policy allows the user to perform five actions (DeleteItem, GetItem, PutItem, Scan, and UpdateItem) on a table 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.

Define a Mapping Class

In DynamoDB, a database is a collection of tables. A table is a collection of items. And each item is a collection of attributes. Let's say that we want to create a bookstore app. For this app, each item in the table represents a book, and each item has four attributes: `id`, `isbn`, `title`, and `hardCover`. (Of course, if we were creating a real bookstore app, we'd need additional attributes to represent author, price, and so forth.) Each item also has a hash key—in this case, `id`—which is the primary key for the table. Each item in DynamoDB will be mapped to a `Book` object in the Java code, and you can directly manipulate the item through the `Book` instance.

To establsh the mappings, DynamoDB defines annotations, including the following:

- `@DynamoDBTable`–Identifies the target table in DynamoDB.
- `@DynamoDBHashKey`–Maps a class property to the hash attribute of the table.
- `@DynamoDBAttribute`–Maps a class property to an item attribute.

For a complete list of the annotations that the Object Mapper offers, see Java: Object Persistence Model. Let's create a mapping class, `Book`:

```
@DynamoDBTable(tableName = "Bookstore")
public static class Book {
    private int id;
    private String isbn;
    private String title;
    private Boolean hardCover;

    @DynamoDBHashKey(attributeName = "id")
    public int getId() {
     return id;
    }

    public void setId(int id) {
     this.id = id;
    }

    @DynamoDBAttribute(attributeName = "isbn")
    public String getIsbn() {
        return isbn;
    }

    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }

    @DynamoDBAttribute(attributeName = "title")
    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    @DynamoDBAttribute(attributeName = "hardCover")
    public Boolean getHardCover() {
        return hardCover;
    }

    public void setHardCover(Boolean hardCover) {
        this.hardCover = hardCover;
    }
}
```

Note that id is primitive, and hardCover is a nullable type. With the DynamoDB Object Mapper, primitives and nullable types behave differently. On a save(), an unset nullable type won't be sent to DynamoDB; an unset primitive will be sent as its default value.

Create a DynamoDB Table

Now we need to create a DynamoDB table for our mapping class. We could create the table in our code or through the console. For this example, we'll use the console.To create a 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 **Number** and enter **id** 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.

Interact with Stored Objects

Now that we have a database, a mapping class, and an Object Mapper, we can use the Object Mapper to write an instance of the Book class to a corresponding item in the Bookstore table. First we'll build the Book object:

```
Book book = new Book();
book.setId(17);
book.setIsbn("222-2222222222");
book.setTitle("Some Title");
book.setHardCover(true);
```

And then we'll use the save() method of DynamoDBMapper to save book to DynamoDB.

```
mapper.save(book);
```

To update an item, just set new attributes and save the object again:

```
book.setTitle("Updated Title");
book.setHardCover(false);
mapper.save(book);
```

Note that setting a new hash key will actually create a new item in the database, even though it doesn't create a new object on the client side. For example, calling book.setId(19) would result in a new item, and the item with an id of 17 would still exist in the database.

Using an object's hash key (in this case, `id`), it's easy to load the corresponding item from the database. The following code snippet returns the Book item with `id` 7:

```
Book anotherBook = mapper.load(Book.class, 7);
```

To delete an item from the database, just use the `delete()` method and pass in the object to be deleted:

```
mapper.delete(anotherBook);
```

Perform a Scan Request

We can also retrieve items from the database with a scan request. A scan request scans the table and returns the results in an undetermined order.

The returned list of items is lazily loaded when possible, so calls to DynamoDB will be made only as needed. When you need to download an entire dataset in advance, you can call a `size()` method on the list to fetch the entire list.

The list returned by the Object Mapper can't be modified, and an attempt to do so will result in an exception. If you want to use the result of a scan as a data source for a modifiable user interface component (for example, an editable `ListActivity`), you'll need to create a modifiable list object and move all of the data to it.

Scan is an expensive operation and should be used with care to avoid disrupting higher priority traffic on the table. Go to the Amazon DynamoDB Developer Guide for more recommendations for safely using the scan operation. The snippet below shows how to scan our example table:

```
DynamoDBScanExpression scanExpression = new DynamoDBScanExpression();
PaginatedScanList<Book> result = mapper.scan(Book.class, scanExpression);
// Do something with result.
```

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 Android provides a simple, high-level client 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 Android into your app and import the necessary libraries. To do so, follow these steps:

1. Download the SDK and unzip it.
2. If you haven't already done so, add the following JARs from `aws-android-sdk-2.x.x\lib\release` to a `libs` folder in your Android project:
   - `aws-android-sdk-x.x.x-core.jar`
   - `aws-android-sdk-x.x.x-kinesis.jar`
3. Make sure that the new libraries are included when building your app. The specific procedure depends on the IDE you're using. If you're working in Eclipse, you can simply drag the JARs into your `libs` folder, and Eclipse will adjust the build path for you.
4. In `AndroidManifest.xml`, set the following permission, if it's not already present:

```
<uses-permission android:name="android.permission.INTERNET" />
```

5. Add the following imports to the main activity of your app.

```
import com.amazonaws.mobileconnectors.kinesisrecorder.*;
import com.amazonaws.auth.CognitoCachingCredentialsProvider;
import com.amazonaws.regions.Regions;
```

6. 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. 12).
7. 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 Kinesis stream 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've imported the necessary libraries and have your credentials object, you can instantiate `KinesisRecorder`. `KinesisRecorder` is a high-level client meant for storing PutRecord requests on an Android device. Storing requests on the device lets you retain data when the device is offline, and it can also increase performance and battery efficiency since the network doesn't need to be awakened as frequently.

Note
`KinesisRecorder` uses synchronous calls, so you shouldn't call `KinesisRecorder` methods on the main thread.

When you create the `KinesisRecorder` client, you'll pass in a directory and an AWS region. The directory should be empty the first time you instantiate `KinesisRecorder`; it should be private to your application; and, to prevent collision, it should be used only by `KinesisRecorder`. The following snippet creates a directory and instantiates the `KinesisRecorder` client, passing in a Cognito credentials object (`cognitoProvider`), a region enum, and the directory.

```
String kinesisDirectory = "YOUR_UNIQUE_DIRECTORY";
KinesisRecorder recorder = new KinesisRecorder(cognitoProvider, Re
gions.US_WEST_2, getDir(kinesisDirectory, 0));
```

You'll use `KinesisRecorder` to save records and then send them in a batch.

```
recorder.saveRecord("MyData".getBytes(),"MyStreamName");
recorder.submitAllRecords();
```

Note
For the `saveRecord()` request above to work, you would have to have created a stream named **MyStreamName**. You can create new streams in the Amazon Kinesis console.

If `submitAllRecords()` is called while the app is online, requests will be sent and removed from the disk. If `submitAllRecords()` is called while the app is offline, requests will be kept on disk until `submitAllRecords()` is called while online. This applies even if you lose your internet connection midway through a submit. So if you save ten requests, call `submitAllRecords()`, send five, and then lose the Internet connection, you have five requests left on disk. These remaining five will be sent the next time `submitAllRecords()` is invoked online.

To see how much space the `KinesisRecorder` client is allowed to use, you can call `getDiskByteLimit()`.

```
Long byteLimit = recorder.getDiskByteLimit();
// Do something with byteLimit.
```

Alternatively, you can retrieve the same information by getting the `KinesisRecorderConfig` object for the recorder and calling `getMaxStorageSize()`:

```
KinesisRecorderConfig kinesisRecorderConfig = recorder.getKinesisRecorderCon
fig();
Long maxStorageSize = kinesisRecorderConfig.getMaxStorageSize();
```

```
// Do something with maxStorageSize.
```

If you exceed the storage limit for `KinesisRecorder`, requests will not be saved or sent. `KinesisRecorderConfig` has a default `maxStorageSize` of 8 MiB. You can configure the maximum allowed storage via the `withMaxStorageSize()` method of `KinesisRecorderConfig`.

To check the number of bytes currently stored in the directory passed in to the `KinesisRecoder` constructor, call `getDiskBytesUsed()`:

```
Long bytesUsed = recorder.getDiskBytesUsed();
// Do something with bytesUsed.
```

To learn more about working with Amazon Kinesis, see Amazon Kinesis Developer Resources. To learn more about the Kinesis classes, see the API Reference for the Android SDK.

Document History

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

Last documentation update: June 30, 2014

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