
Login with Amazon Getting Started Guide for Android apps

Login with 

Login with Amazon: Getting Started Guide for Android

Copyright © 2017 Amazon.com, Inc., or its affiliates. All rights reserved.

Amazon and the Amazon logo are trademarks of Amazon.com, Inc. or its affiliates. All other trademarks not owned by Amazon are the property of their respective owners.

Contents

Introduction	2
Install the Android Developer Tools.....	3
Install the Login with Amazon SDK for Android	3
Run the Sample App.....	3
Register with Login with Amazon.....	4
Register Your Login with Amazon Application	4
Add Android Settings to your Application	5
Android App Signatures and API Keys.....	8
Determine the Android App Signature	9
Retrieve the Android API Key.....	9
Create a Login with Amazon Project	10
Create a New Login with Amazon Project.....	10
Install the Login with Amazon Library.....	10
Set Network Permissions for Your App	10
Add Your API Key to Your Project	11
Handle Configuration Changes for Your Activity	12
Add a WorkflowActivity to your Project	12
Add a Login with Amazon Button to Your App	13
Use the SDK for Android APIs	14
Handle the Login Button and Authorize the User.....	14
Fetch User Profile Data	17
Check for User Login at Startup	17
Clear Authorization Data and Log Out a User	18

Introduction

Topics

- [Install the Android Developer Tools](#)
- [Register with Login with Amazon](#)
- [Create a Login with Amazon Project](#)
- [Add a Login with Amazon Button to your app](#)
- [Use the SDK for Android APIs](#)

In this guide we will show you how to add Login with Amazon to your Android app, using the Login with Amazon SDK for Android v3.0+. You can also use this SDK to add Login with Amazon to your Amazon Fire-built apps.

After completing this guide you should have a working Login with Amazon button in your app that allows users to login with their Amazon credentials. To learn more about the login flow your customers will experience when they use Login with Amazon within your app, please see our [Customer Experience Overview for Android/Fire apps](#).

Install the Android Developer Tools

The Login with Amazon SDK for Android will help you add Login with Amazon to your Android, Fire TV, and Fire Tablet applications. We recommend you use the Login with Amazon SDK for Android with Android Studio. For steps on how to install Android Studio and on getting the Android SDK set up, see [Get the Android SDK](#) on developer.android.com.

To use the Login with Amazon SDK for Android, your Android application must meet one of these minimum requirements:

- Minimum SDK Version (minSdkVersion) of Android 3.0 (API Level 11) or higher.
- Minimum SDK Version (minSdkVersion) of Android 2.2 (API Level 8) or higher with the v4 [Android Support Library](#).

When the Android SDK is installed, find the **SDK Manager** application in your Android installation. To develop for Login with Amazon, you must use the SDK Manager to install the minimum SDK requirements above. See [Adding SDK Packages](#) on developer.android.com for more information on using SDK Manager.

After installing the SDK, set up an Android Virtual Device (AVD) for running your apps. See [Managing Virtual Devices](#) on developer.android.com for instructions on setting up a virtual device.

When your development environment is set up, you can [Install the Login with Amazon SDK for Android](#) or [Run the Sample App](#), as described below.

Install the Login with Amazon SDK for Android

The Login with Amazon SDK for Android comes in two packages. The first contains the Android library and supporting documentation. The second contains a sample application that allows a user to login and displays their profile data.

If you have not already installed the Android SDK or the Android Development Tools, see the [Installing the Android Developer Tools](#) section above.

1. Download [LoginWithAmazonSDKForAndroid.zip](#) and extract the files to a directory on your hard drive. You should see a **docs** and a **lib** subdirectory.
2. Open **docs/index.html** to view the Login with Amazon Android API Reference.
3. See [Install the Login with Amazon Library](#) for instructions on how to add the library and documentation to an Android project.

When the Login with Amazon SDK for Android is installed, you can [Create a New Login with Amazon Project](#) after you [Register with Login with Amazon](#).

Run the Sample App

To run the sample application, import the sample into an AndroidStudio workspace.

1. Download [SampleLoginWithAmazonAppForAndroid-src.zip](#) and extract the files to a directory on your hard drive.
2. Start AndroidStudio and select **Open an existing Android Studio project**.
3. Browse to the **SampleLoginWithAmazonApp** directory obtained after extracting the downloaded zip file in Step 1.
4. From the **Build** menu, click **Make Project**, and wait for the project to finish building.

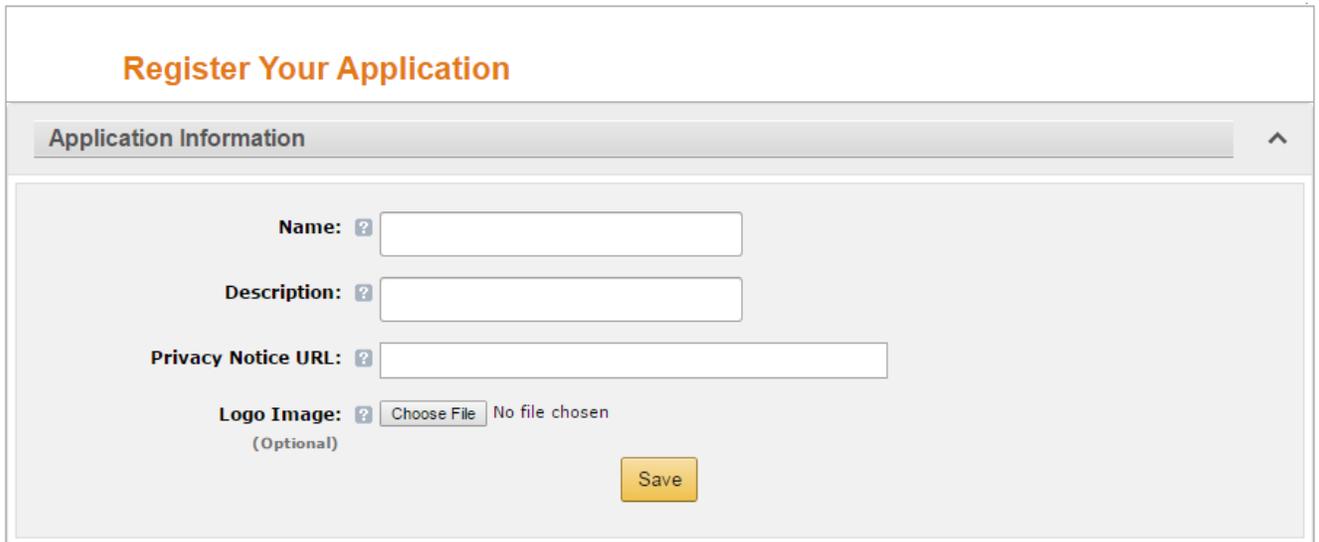
5. From the **Run** menu, click **Run** and then click the **SampleLoginWithAmazonApp**.
6. Select the emulator or connected Android device and click **Run**.

Register with Login with Amazon

Before you can use Login with Amazon on a website or in a mobile app, you must register an application with Login with Amazon. Your Login with Amazon application is the registration that contains basic information about your business, and information about each website or mobile app you create that supports Login with Amazon. This business information is displayed to users each time they use Login with Amazon on your website or mobile app. Users will see the name of your application, your logo, and a link to your privacy policy. These steps demonstrate how to register your Android app for use with Login with Amazon.

Register Your Login with Amazon Application

1. Go to <https://login.amazon.com>.
2. If you have signed up for Login with Amazon before, click **App Console**. Otherwise, click **Sign Up**. You will be redirected to Seller Central, which handles application registration for Login with Amazon. If this is your first time using Seller Central, you will be asked to set up a Seller Central account.
3. Click **Register New Application**. The **Register Your Application** form will appear:



The screenshot shows a web form titled "Register Your Application". At the top, there is a tab labeled "Application Information". Below the tab, there are four input fields, each with a question mark icon to its left:

- Name:** A text input field.
- Description:** A text input field.
- Privacy Notice URL:** A text input field.
- Logo Image:** A text input field with a "Choose File" button and the text "No file chosen" to its right. Below this field is the text "(Optional)".

At the bottom right of the form, there is a yellow "Save" button.

- a. In the **Register Your Application** form, enter a **Name** and a **Description** for your application. The **Name** is the name displayed on the consent screen when users agree to share information with your application. This name applies to Android, iOS, and website versions of your application. The **Description** helps you differentiate each of your Login with Amazon applications and is not displayed to users.
- b. Enter a **Privacy Notice URL** for your application. The **Privacy Notice URL** is the location of your company's or application's privacy policy (for example, <http://www.example.com/privacy.html>). This link is displayed to users on the consent screen.
- c. If you want to add a **Logo Image** for your application, click **Browse** and locate the applicable image.

This logo is displayed on the sign-in and consent screen to represent your business or website. The logo will be shrunk to 50 pixels in height if it is taller than 50 pixels; there is no limitation on the width of the logo.

4. Click **Save**. Your sample registration should look similar to this:



App ID: amzn1.application.44ebe3ee3aef41e5acff0e8aee2ee55f

The screenshot shows the "Settings" tab of an application configuration page. At the top, there are two tabs: "Settings" (selected) and "Metrics". Below the tabs is a section titled "Application Information" with an upward-pointing arrow. This section contains four fields: "Name" (Zappos.com), "Description" (Powered by Service), "Privacy Notice URL" (http://www.zappos.com/privacy-policy), and "Logo Image" (Zappos.com logo, marked as optional). An "Edit" button is located at the bottom right of this section. Below the "Application Information" section are three expandable sections: "Web Settings", "Android Settings", and "iOS Settings", each with a plus icon and a downward-pointing arrow.

After your basic application settings are saved, you can add settings for specific websites and mobile apps that will use this Login with Amazon account.

Add Android Settings to your Application

To register an Android App, you have the choice of registering an app through the Amazon Appstore ([Add an Android App for Amazon Appstore](#)) or directly with Login with Amazon ([Add an Android App without Appstore](#)). When your app is registered, you will have access to an API key that will grant your app access to the Login with Amazon authorization service.

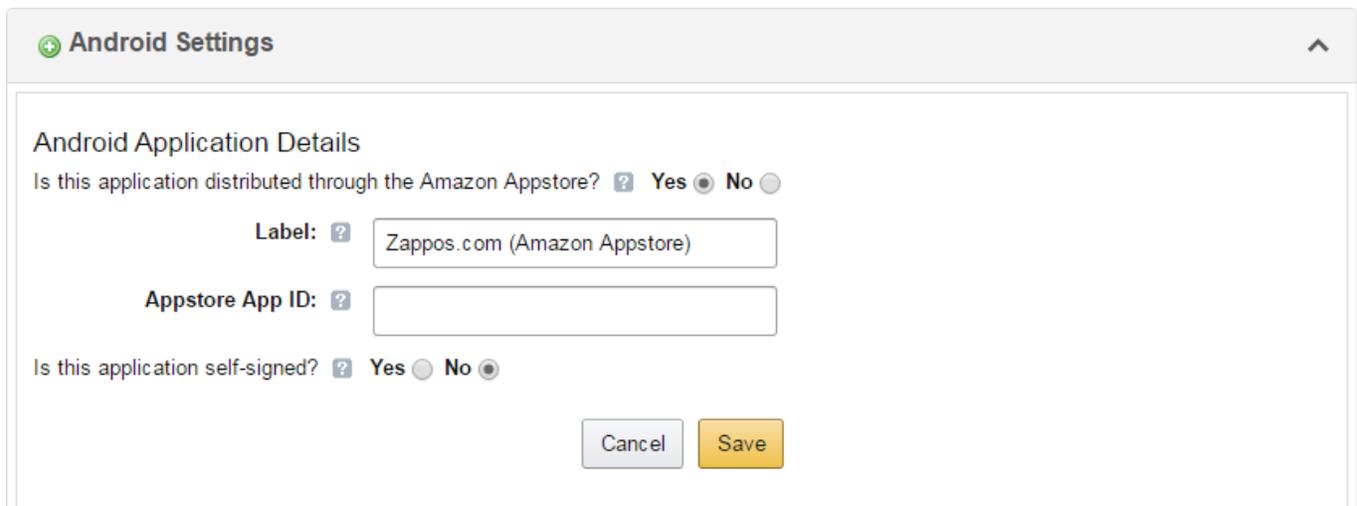
Note: If you plan to use Amazon Device Messaging within your Android app, please contact lwa-support@amazon.com with:

- The email address of the Amazon account you used to sign up for Login with Amazon.
- The email address of the Amazon account you used to sign up for the Amazon Appstore (if different).
- The name on your Seller Central account (in Seller Central, click **Settings > Account Info > Seller Information**, and use the **Display Name**).
- The name on your Amazon Appstore developer account (in the Mobile App Distribution site, click **Settings > Company Profile** and use the **Developer Name or CompanyName**).

Add an Android app for Amazon Appstore

The following steps will add an Amazon Appstore app to your Login with Amazon account:

1. From the Application screen, click **Android Settings**. If you already have an Android app registered, look for the **Add API Key** button in the **Android Settings** section. The **Android Application Details** form will appear:



The screenshot shows a dialog box titled "Android Settings" with a sub-section "Android Application Details". It contains the following elements:

- A question: "Is this application distributed through the Amazon Appstore?" with radio buttons for "Yes" (selected) and "No".
- A "Label:" field with a question mark icon and a text input containing "Zappos.com (Amazon Appstore)".
- An "Appstore App ID:" field with a question mark icon and an empty text input.
- A second question: "Is this application self-signed?" with radio buttons for "Yes" and "No" (selected).
- At the bottom, there are "Cancel" and "Save" buttons.

2. Select **Yes** in answer to the question "Is this application distributed through the Amazon Appstore?"
3. Enter the **Label** of your Android App. This does not have to be the official name of your app. It simply identifies this particular Android app among the apps and websites registered to your Login with Amazon application.
4. Add your **Amazon Appstore ID**.
5. If you self-signed your app, add self-signing information. This will allow you to obtain an API key during development without using the Appstore directly.
 - a. If your app is not being signed through the Amazon Appstore, select **Yes** in answer to the question "Is this application self-signed?"The **Android Application Details** form will expand:

- b. Enter your **Package Name**.

This must match the package name of your Android project. To determine the package name of your Android Project, open the project in your choice of Android developer tool. Open **AndroidManifest.XML** in Package Explorer and select the **Manifest** tab. The first entry is the **Package** name.

- c. Enter the app **Signature**.

This is a SHA-256 hash value used to verify your application. The signature must be in the form of 32 hexadecimal pairs separated by colons (for example:

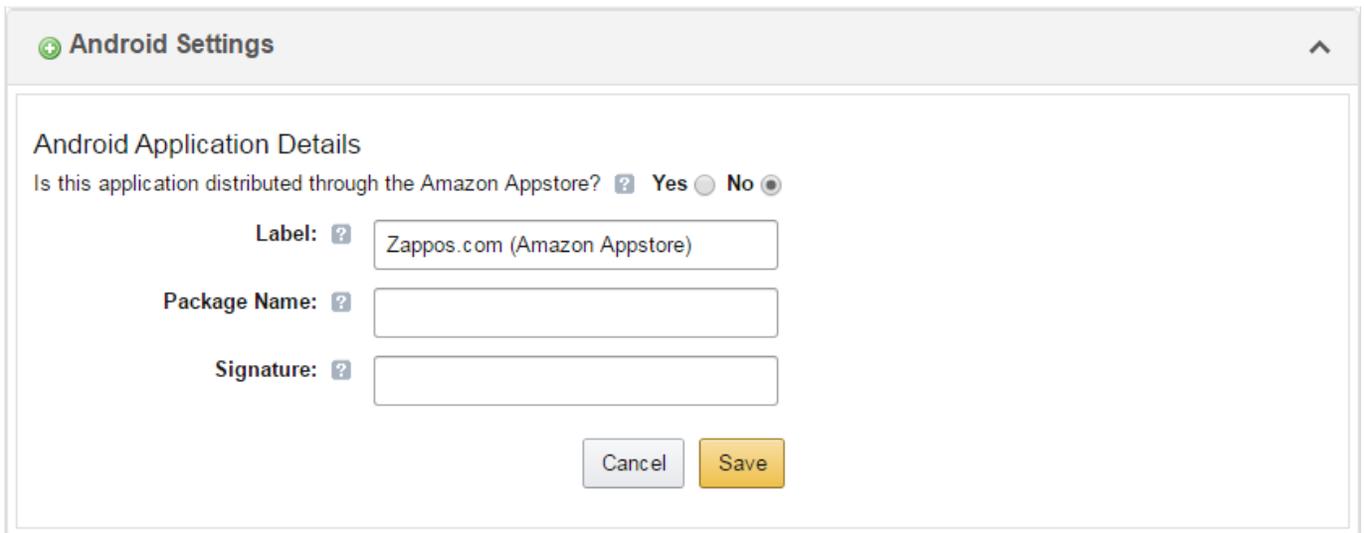
01:23:45:67:89:ab:cd:ef:01:23:45:67:89:ab:cd:ef:01:23:45:67:89:ab:cd:ef:01:23:45:67:89:ab:cd:ef). See [Android App Signatures and API Keys](#) for steps you can use to extract the signature from your project.

6. Click **Save**.

Add an Android App without Appstore

If you wish to register your Android app without using the Amazon Appstore, you can use the following steps to register your Package Name and Signature with Login with Amazon:

1. From the Application screen, click **Android Settings**. If you already have an Android app registered, look for the **Add API Key** button in the **Android Settings** section. The **Android Application Details** form will appear:



2. Select **No** in answer to the question "Is this application distributed through the Amazon Appstore?"
3. Enter the **Label** of your Android App.
This does not have to be the official name of your app. It simply identifies this particular Android app among the apps and websites you have registered.
4. Enter your **Package Name**. This must match the package name of your Android project.
To determine the package name of your Android Project, open the project in your choice of Android developer tool. Open **AndroidManifest.XML** in Package Explorer and select the **Manifest** tab. The first entry is the **Package** name.
5. Enter the app **Signature**.
This is a SHA-256 hash value used to verify your application. The signature must be in the form of 32 hexadecimal pairs separated by colons (for example:
01:23:45:67:89:ab:cd:ef:01:23:45:67:89:ab:cd:ef:01:23:45:67:89:ab:cd:ef:01:23:45:67:89:ab:cd:ef). See the [Android App Signatures and API Keys](#) section below for steps you can use to extract the signature from your project.
6. Click **Save**.

If different versions of your app have different signatures or package names, such as for one or more testing versions and a production version, each version requires its own API Key. From the **Android Settings** of your app, click the **Add API Key** button to create additional keys for your app (one per version).

Android App Signatures and API Keys

The app signature is a SHA-256 hash value that is applied to every Android app when it is built. Amazon uses the app signature to construct your API Key. The API Key enables Amazon services to recognize your app. If you use the Amazon Appstore to sign your app, the API key is provided automatically. If you are not using the Amazon Appstore, you will need to manage your API key manually.

App signatures are stored in a keystore. Generally for Android apps there is a debug keystore and a release keystore. To find the location of the debug keystore in Android Studio, open the **Build** menu, select **Edit Build Types**, then go to the **Signing** tab, and locate the debug keystore in the **Store File** field.

A release keystore is normally created when you export your Android app to create a signed APK file. Through the export process, if you are creating a new release keystore you will select its location. By default it will be placed in the same location as your default debug keystore.

If you have registered your app using the debug signature during development, you will have to add a new Android setting to your application when you are ready to release the app. The new app setting must use the signature from the release keystore.

See [Signing Your Applications](https://developer.android.com/signing/signing-your-applications) on developer.android.com for more information.

Determine the Android App Signature

1. If you have a signed APK file:
 - a. Unzip the APK file and extract **CERT.RSA**. (You can rename the APK extension to ZIP if necessary).
 - b. From the command line, run:

```
keytool -printcert -file CERT.RSA
```

`keytool` is located in the **bin** directory of your Java installation.

2. If you have a keystore file:
 - a. From the command line, run:

```
keytool -list -v -alias <alias> -keystore <keystore.filename>
```

`keytool` is located in the **bin** directory of your Java installation. The alias is the name of the key used to sign the app.

- b. Enter the password for the key and press **Enter**.
3. Under **Certificate Fingerprints**, copy the **SHA256** value.

Retrieve the Android API Key

When you have registered an Android setting and provided an app signature, you can retrieve the API key from the registration page for your Login with Amazon application. You will need to place that API key into a file in your Android project. Until you do, the app will not be authorized to communicate with the Login with Amazon authorization service.

1. Go to <https://login.amazon.com>.
2. Click **App Console**.
3. In the **Applications** box at left, select your application.
4. Find your Android app under the **Android Settings** section (If you have not yet registered an Android app, see [Add an Android App for Amazon Appstore](#)).
5. Click **Generate API Key Value**. A popup window will display your API key. To copy the key, click **Select All** to select the entire key.

Note: The API Key Value is based, in part, on the time it is generated. Thus, subsequent API Key Value(s) you generate may differ from the original. You can use any of these API Key Values in your app as they are all valid.
6. See [Add Your API Key to Your Project](#) for instructions on adding the API key to your Android app.

Create a Login with Amazon Project

In this section, you will learn how to create a new Android project for Login with Amazon, configure the project, and add code to the project to sign in a user with Login with Amazon. We will be describing the steps for Android Studio, but you can apply analogous steps to any IDE or Android development tool of your choice.

This guide requires an understanding of *Activities* - a key concept of Android application development. Learn more about [Activities](#) and [Activity Fragments](#) on developer.android.com.

Create a New Login with Amazon Project

If you do not yet have an app project for using Login with Amazon, follow the instructions below to create one. If you have an existing app, skip to [Install the Login with Amazon Library](#).

1. Launch **Android Studio**.
2. From the **File** menu, select **New** and **Project**.
3. Enter an **Application Name** and **Company Name** for your app.
4. Enter the **Application** and **Company Name** corresponding to the package name that you chose when you registered your app with Login with Amazon.
If you haven't registered your app yet, choose a **Package Name** and then follow the instructions in the [Registering with Login with Amazon](#) section after you create your project. If the package name of your app does not match the registered package name, your Login with Amazon calls will not succeed.
5. Select a **Minimum Required SDK** of API 11: Android 3.0 (Honeycomb) or higher, and click **Next**. You can alternatively use a **Minimum Required SDK** of API 8: Android 2.2 (Froyo) or higher when using the v4 [Android Support Library](#).
6. Select the type of activity you want to create and click **Next**.
7. Fill in the relevant details and click **Finish**.

You will now have a new project in your workspace that you can use to call Login with Amazon.

Install the Login with Amazon Library

If you have not yet downloaded the Login with Amazon SDK for Android, see [Install the Login with Amazon SDK for Android](#).

1. Using the file system on your computer, find the **login-with-amazon-sdk.jar** file within the Login with Amazon SDK for Android. Copy it to the clipboard.
2. With your project open in Android Studio, open the **Project View**.
3. Right-click on the parent directory for your project/app in the **Project View** and select **Paste**.
4. Right-click **login-with-amazon-sdk.jar** in the **Project View** and select **Add As Library**.

Set Network Permissions for Your App

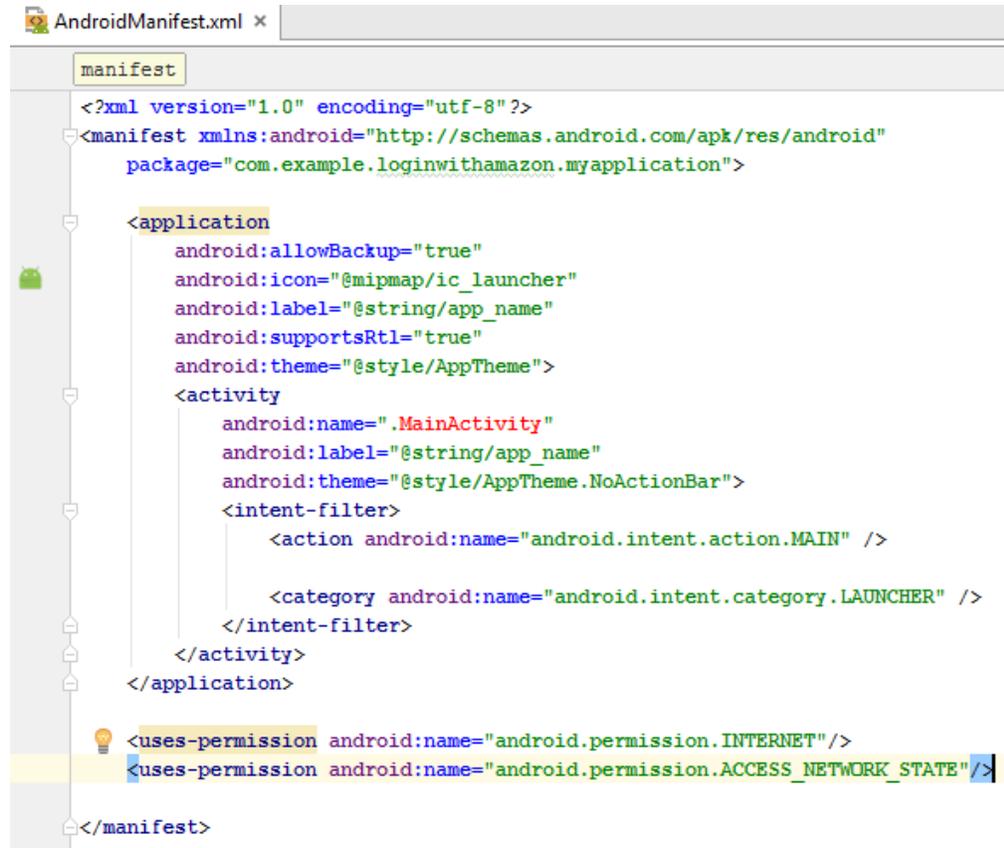
In order for your app to use Login with Amazon, it must access the Internet and access network state information. Your app must assert these permissions in your Android manifest, if it doesn't already.

1. From the **Project View**, double-click **AndroidManifest.xml** to open it.
2. Copy the lines of code displayed below and paste them into the **AndroidManifest.xml** file,

outside of the application block:

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

Example:



```
<?xml version="1.0" encoding="utf-8" ?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.loginwithamazon.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

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

</manifest>
```

Add Your API Key to Your Project

When you register your Android application with Login with Amazon, you are assigned an API key. This is an identifier that the Amazon Authorization Manager will use to identify your application to the Login with Amazon authorization service. If you are using the Amazon Appstore to sign your app, the Appstore will provide the API key automatically. If you are not using the Amazon Appstore, the Amazon Authorization Manager loads this value at runtime from the **api_key.txt** file in the **assets** directory.

1. If you do not have your API Key yet, follow the instructions in [Retrieve the Android API Key](#).
2. From the **Project View** in Android Studio, right-click the **assets** folder, then click **New** and select **File**. If you don't have an **assets** folder, right-click the parent directory for your project, then select **New, Folder, Assets Folder**.
3. Name the file **api_key.txt**.
4. You should now have an editor window for a text file named **api_key.txt**. Add your API key to the text file.
5. In the **File** menu, click **Save**.

Note: If a text editor adds extra characters to your **api_key.txt** file (such as a Byte Order Mark), you may see **ERROR_ACCESS_DENIED** when you try to connect to the Login with Amazon authorization service. If this

occurs, try removing any leading or trailing spaces, line feeds, or suspicious characters. (For example, an editor using Byte Order Mark might add 0xEF 0xBB 0xBF or other hexadecimal sequences to the start of your `api_key.txt` file). You can also try retrieving a new API key.

Handle Configuration Changes for Your Activity

If a user changes the screen orientation or changes the keyboard state of the device while they are logging in, it will prompt a restart of the current activity. This restart will dismiss the login screen unexpectedly. To prevent this, you should set the activity that uses the `authorize` method to handle those configuration changes manually. This will prevent a restart of the activity.

1. In **Package Explorer**, double-click **AndroidManifest.xml**.
2. In the **Application** section, locate the activity that will handle Login with Amazon (for example, **MainActivity**).
3. Add the following attribute to the activity you located in Step 2:

```
android:configChanges="keyboard|keyboardHidden|orientation"
```

Or for API 13 or greater:

```
android:configChanges="keyboard|keyboardHidden|orientation|screenSize"
```

4. From the **File** menu, click **Save**.

Now, when a keyboard or device orientation change happens, Android will call the `onConfigurationChanged` method for your activity. You do not need to implement this function unless there is an aspect of these configuration changes that you want to handle for your app.

Add a WorkflowActivity to your Project

When the user clicks the Login with Amazon button, the API will launch a web browser to present a login and consent page to the user. In order for this browser activity to work, you must add the `WorkflowActivity` to your manifest.

If you have previously integrated with the Login with Amazon SDK or you have the `com.amazon.identity.auth.device.authorization.AuthorizationActivity` activity declared in your `AndroidManifest.xml`, it must be removed and replaced with the `WorkflowActivity`.

1. In **Package Explorer**, double-click **AndroidManifest.xml**.
2. In the **Application** section, add the following code.

```

<activity
  android:name=
  "com.amazon.identity.auth.device.workflow.WorkflowActivity"
  android:theme="@android:style/Theme.NoDisplay"
  android:allowTaskReparenting="true"
  android:launchMode="singleTask">
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    <!-- android:host must use the full package name
         found in Manifest General Attributes -->
    <data
      android:host="${applicationId}"
      android:scheme="amzn" />
  </intent-filter>
</activity>

```

Note: If you are not using the Gradle build system, replace `${applicationId}` with your package name for this app.

Add a Login with Amazon Button to Your App

Login with Amazon provides several standard buttons that you can use to prompt users to login from your app. This section gives steps for downloading an official Login with Amazon image and pairing it with an Android `ImageButton`.

1. Add a standard `ImageButton` to your app.

For more information on Android buttons and the `ImageButton` class, see [Buttons](#) on developer.android.com.

2. Give your button an id.

In the button XML declaration, set the `android:id` attribute to `@+id/login_with_amazon`. For example:

```
android:id="@+id/login_with_amazon"
```

3. Choose a button image.

Consult our Login with Amazon [Style Guidelines](#) for a list of buttons that you can use in your app. Download a copy of the **LWA_Android.zip** file. Extract a copy of your preferred button for each screen density your app supports (xxhdpi, xhdpi, hdpi, mdpi, or tvdpi). For more information on supporting multiple screen densities in Android, see [Alternative Layouts](#) in the “Supporting Multiple Screens” topic on developer.android.com.

4. Copy the appropriate button image files to your project.

For each screen density that you support (xhdpi, hdpi, mdpi, or ldpi), copy the downloaded button to the **res/drawable** directory for that screen density.

5. Declare the button image.

In the button XML declaration, set the `android:src` attribute to the name of the button you have chosen. For example:

```
android:src="@drawable/btnlwa_gold_loginwithamazon.png"
```

6. Load your app, and verify that the button now has a Login with Amazon image. You should verify that the button displays properly for each screen density you support.

Use the SDK for Android APIs

In this section, you will add code to your project to sign in a user with Login with Amazon.

Handle the Login Button and Authorize the User

This section explains how to call the `authorize` API to login a user. This includes creating an `onClick` listener for your Login with Amazon button in the `onCreate` method of your app.

1. Add Login with Amazon to your Android project. See [Install the Login with Amazon Library](#).

2. Initialize `RequestContext`.

You will need to declare a `RequestContext` variable and create a new instance of the class. The best place to initialize `RequestContext` is in the `onCreate` method of your Android activity or fragment. For example:

```
private RequestContext requestContext;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    requestContext = RequestContext.create(this);
}
```

3. Create an `AuthorizeListener`.

`AuthorizeListener` will process the result of the `authorize` call. It contains three methods: `onSuccess`, `onError`, and `onCancel`. Create the `AuthorizeListener` interface in-line with a `registerListener` call in the `onCreate` method of your Android activity or fragment.

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    requestContext = RequestContext.create(this);

    requestContext.registerListener(new AuthorizeListener() {

        /* Authorization was completed successfully. */
        @Override
        public void onSuccess(AuthorizeResult result) {
            /* Your app is now authorized for the requested scopes */
        }

        /* There was an error during the attempt to authorize the
           application. */
        @Override
        public void onError(AuthError ae) {
            /* Inform the user of the error */
        }

        /* Authorization was cancelled before it could be completed. */
        @Override
        public void onCancel(AuthCancellation cancellation) {
            /* Reset the UI to a ready-to-login state */
        }
    });
}

```

Note: If you're using a fragment and capturing references to `View` objects in your `AuthorizeListener` implementation, create `AuthorizeListener` in the [onCreateView](#) method instead of `onCreate`. This ensures the `View` object references are set when the call to `authorize` finishes.

4. Implement `onSuccess`, `onError`, and `onCancel` for your `AuthorizeListener`. Because the authorization process presents a login screen (and possibly a consent screen) to the user in a web browser (or a `WebView`), the user will have an opportunity to cancel the login or navigate away. If they explicitly cancel the login process, `onCancel` is called, and you will want to reset your user interface.

If the user navigates away from the login screen in the browser or `WebView`, then switches back to your app, the SDK will not detect that the login was not completed. If you detect user activity in your app before login is completed, you can assume they have navigated away from the browser and react accordingly.

5. Call `RequestContext.onResume`. In order to accommodate the Android application lifecycle, implement the `onResume` method in your activity or fragment. This will trigger all listeners registered with `registerListener` in the event that your app is closed by the operating system before the user completes an authorization flow.

```

@Override
protected void onResume() {
    super.onResume();
    requestContext.onResume();
}

```

6. Call `AuthorizationManager.authorize`.

In the `onClick` handler for your Login with Amazon button, call `authorize` to prompt the user to login and authorize your application.

This method will enable the user to sign in and consent to the requested information in one of the following ways:

1. Switches to the system browser
2. Switches to `WebView` in a secure context (if the Amazon Shopping app is installed to the device)

The secure context for the second option is available when the Amazon Shopping app is installed to the device. Amazon-created devices running Fire OS (for example Kindle Fire, Fire Phone, and Fire TV) always use this option even if there is no Amazon Shopping app on the device. Because of this, if the user is already signed in to the Amazon Shopping app, this API will skip the sign in page, leading to a **Single Sign-On** experience for the user. See [Customer Experience in Android/Fire](#) apps to learn more.

When your application is authorized, it is authorized for one or more data sets known as **scopes**. A scope encompasses the user data you are requesting from Login with Amazon. The first time a user logs in to your app, they will be presented with a list of the data you are requesting and asked for approval.

Login with Amazon currently supports the following scopes:

Scope name	Description
<code>profile</code>	Gives access to the user's name, email address, and Amazon account ID.
<code>profile:user_id</code>	Gives access to the user's Amazon account ID only.
<code>postal_code</code>	Gives access to the user's zip/postal code on file for their Amazon account.

`AuthorizationManager.authorize` is an asynchronous call, so you do not have to block the UI thread or create a worker thread of your own. To call `authorize`, pass an `AuthorizeRequest` object that can be built using `AuthorizeRequest.Builder`:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    /* Previous onCreate declarations omitted */

    // Find the button with the login_with_amazon ID
    // and set up a click handler
    View loginButton = findViewById(R.id.login_with_amazon);
    loginButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            AuthorizationManager.authorize(new AuthorizeRequest
                .Builder(requestContext)
                .addScopes(ProfileScope.profile(), ProfileScope.postalCode())
                .build());
        }
    });
}
```

Fetch User Profile Data

This section explains how to use the `User` API to retrieve a user's profile data after they've been authorized. The profile data you can retrieve is based on the scope indicated in the `authorize:withHandler:` call.

1. Call `User.fetch`.

`User.fetch` returns the user's profile data to you through the `Listener<User, AuthError>` callback. `Listener<User, AuthError>` contains two methods: `onSuccess` and `onError` (it does not support `onCancel` because there is no way to cancel a `User.fetch` call). `onSuccess` receives a `User` object with profile data, while `onError` receives an `AuthError` object with information on the error. `updateProfileData` is an example of a function your app could implement to display profile data in the user interface.

```
private void fetchUserProfile() {
    User.fetch(this, new Listener<User, AuthError>() {

        /* fetch completed successfully. */
        @Override
        public void onSuccess(User user) {
            final String name =
                user.getUserName();
            final String email = user.getUserEmail();
            final String account = user.getUserId();
            final String zipcode =
                user.getUserPostalCode();

            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    updateProfileData(name, email, account, zipcode);
                }
            });
        }

        /* There was an error during the attempt to get the profile. */
        @Override
        public void onError(AuthError ae) {
            /* Retry or inform the user of the error */
        }
    });
}
```

Note: `User.getUserPostalCode` is only returned if you request the `ProfileScope.postalCode()` scope.

Check for User Login at Startup

If a user logs into your app, closes the app, and restarts the app later, the app is still authorized to retrieve data. The user is not logged out automatically. At startup, you can show the user as logged in if your app is still authorized. This section explains how to use `getToken` to see if the app is still authorized.

1. Call `getToken`.

In the `onStart` method of your activity or fragment, call `getToken` to see if the

application is still authorized. `getToken` retrieves the raw access token that the `AuthorizationManager` uses to access a user profile. If the token value is not `null`, then the app is still authorized and you can proceed to [fetch user profile data](#). `getToken` requires the same scopes you requested in your call to `authorize`.

`getToken` supports asynchronous calls in the same manner as `User.fetch`, so you do not have to block the UI thread or create a worker thread of your own. To call `getToken` asynchronously, pass an object that supports the `Listener<AuthorizeRequest, AuthError>` interface as the last parameter.

2. Declare a `Listener<AuthorizeResult, AuthError>`.
Your implementation of the `Listener<AuthorizeResult, AuthError>` interface processes the result of the `getToken` call. `Listener` contains two methods: `onSuccess` and `onError` (it does not support `onCancel` because there is no way to cancel a `getToken` call).
3. Implement `onSuccess` and `onError` for your `Listener<AuthorizeResult, AuthError>`.
`onSuccess` receives an `AuthorizeResult` object with an access token, while `onError` receives an `AuthError` object with information on the error.

```
@Override
protected void onStart() {
    super.onStart();
    Scope[] scopes = { ProfileScope.profile(),
        ProfileScope.postalCode() };
    AuthorizationManager.getToken(this, scopes, new
        Listener<AuthorizeResult, AuthError>() {

        @Override
        public void onSuccess(AuthorizeResult result) {
            if (result.getAccessToken() != null) {
                /* The user is signed in */
            } else {
                /* The user is not signed in */
            }
        }

        @Override
        public void onError(AuthError ae) {
            /* The user is not signed in */
        }
    });
}
```

Clear Authorization Data and Log Out a User

This section explains how to use the `signOut` method to log the user out of your app. The user will have to login again in order for the app to retrieve profile data. Use this method to log out a user, or to troubleshoot login problems in the app.

1. Implement a logout mechanism.
When a user has successfully logged in, you should provide a logout mechanism so they can

clear their profile data and previously authorized scopes. Your mechanism might be a hyperlink, button, or a menu item. For this example, we will create an `onClick` method for a button.

2. Call `signOut`.
Call `signOut` in your logout handler to remove a user's authorization data (access tokens, profile) from the local store. `signOut` takes an [Android context](#) and a `Listener<Void, AuthError>` to handle success or failure.
3. Declare an anonymous `Listener<Void, AuthError>`.
Your implementation of `Listener<Void, AuthError>` processes the result of the `signOut` call. Anonymous classes are useful for capturing variables from the enclosing scope. See [Handle the Login Button and Authorize the User](#) for an example that declares listener classes.
4. Implement `onSuccess` and `onError` for your `Listener<Void, AuthError>`.
When `signOut` succeeds you should update your UI to remove references to the user, and provide a login mechanism users can use to login again. If `signOut` returns an error, you can let the user try to log out again.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    /* Previous onCreate declarations omitted */

    // Find the button with the logout ID and set up a click handler
    View logoutButton = findViewById(R.id.logout);
    logoutButton.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View v) {
            AuthorizationManager.signOut(getApplicationContext(), new
            Listener<Void, AuthError>() {
                @Override
                public void onSuccess(Void response) {
                    // Set logged out state in UI
                }
                @Override
                public void onError(AuthError authError) {
                    // Log the error
                }
            });
        }
    });
}
```