OneLogin NAPPS SDK for Android

www.onelogin.com | **twitter.com/onelogin**
OneLogin, Inc. | 150 Spear Street, Suite 1400, San Francisco, CA 94105

855.426.7227

Sidebar text:

HEADING 1
CONTENT CONTENT CONTENT CONTENT CONTENT CONTENT CONTENT CONTENT CONTENT CONTENT
HEADING 2
CONTENT CONTENT CONTENT CONTENT CONTENT CONTENT CONTENT CONTENT CONTENT
HEADING 3
CONTENT CONTENT CONTENT CONTENT CONTENT CONTENT CONTENT CONTENT CONTENT CONTENT

OVERVIEW

Welcome to OneLogin's NAPPS software development kit. Contained within, you will find a few elements that will allow you develop and test the functionality of the Token Agent.

In the typical use-case, after successfully authenticating the a user in question, the Token Agent obtains additional security tokens representing that specific user from an identity server for other native applications. Our testing version contains a 'Mock Agent' that doesn't connect to any actual server or outside service, but operates as if one was successfully present, and will allow the testing suite to run successfully.

What you're getting with this testing suite:

- The Mock Token Agent
- The Sample Application
- An SDK Library for the test application to run on.

You'll also be getting the full source code for both the Mock Agent and the Sample Application, with lines of both contained on the next few pages.

BEST PRACTICES: GETTING STARTED

It is highly recommended that you build out your sample application with the Mock Token Agent to ensure that all binding connectors are properly configured. Once you have established the correct usage flows, remove the mock token agent and install the the launcher on your device which should leverage the pre-configured settings established earlier.

Currently, the mobile NAPPs application connectors are not publicly available, so please contact napps-support@onelogin.com to have your connector configured.

CREATING A CLIENT

Begin by registering a custom URL scheme for your client application. As shown below, this URL scheme will be used in the token agent as part of the callback function to your application. Note that your URL scheme will not match the example but will resemble the example shown.

```
<activity
        android:screenOrientation="portrait"
        android:name="onelogin.com.signal.Signal"
        android:label="@string/app_name" >

        <intent-filter>
           <action android:name="android.intent.action.MAIN" />

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

        <intent-filter>
           <action android:name="android.intent.action.VIEW" />
           <category android:name="android.intent.category.DEFAULT" />
           <category android:name="android.intent.category.BROWSABLE" />
           <data android:scheme="http" android:host="onelogin.com.signal.Signal" />
        </intent-filter>

     </activity>
```

*The **Test Token Agent** works exactly as the real token agent, except that it doesn't call any actual server and the token retrieved is just a random string.*

The result is that within the Mock Token Agent, the developer can trigger the different error codes from the Settings screen. If any error is sent, the Mock Token Agent won't sent the token and will send the error code instead.

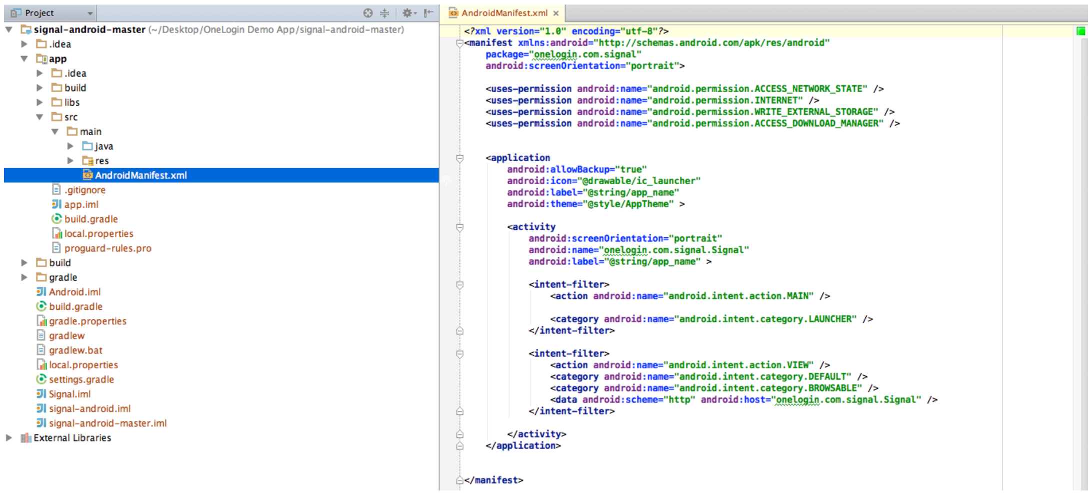
The SDK on the third-party app automatically decodes this error and passes it to the developer.

MAKING A SECONDARY TOKEN REQUEST

With the client created, you will need to insert the Token Request Code found within the Test Request Endpoint into your own codebase. An example of this code will be found between lines 25 to 35 of the Test Request Endpoint and allows for the client to successfully make the second token request from the Token Agent.

FINDING THE CUSTOM URL AND INSTALLING THE .JAR FILE IN ANDROID STUDIO

. After importing the project into Android Studio, finding the required custom URL scheme is fairly straightforward. What you'll be looking for is a filed called **AndroidManifest.xml**, found under the directory path of **app > src > main > AndroidManifest.xml**. Opening the file will display the following:

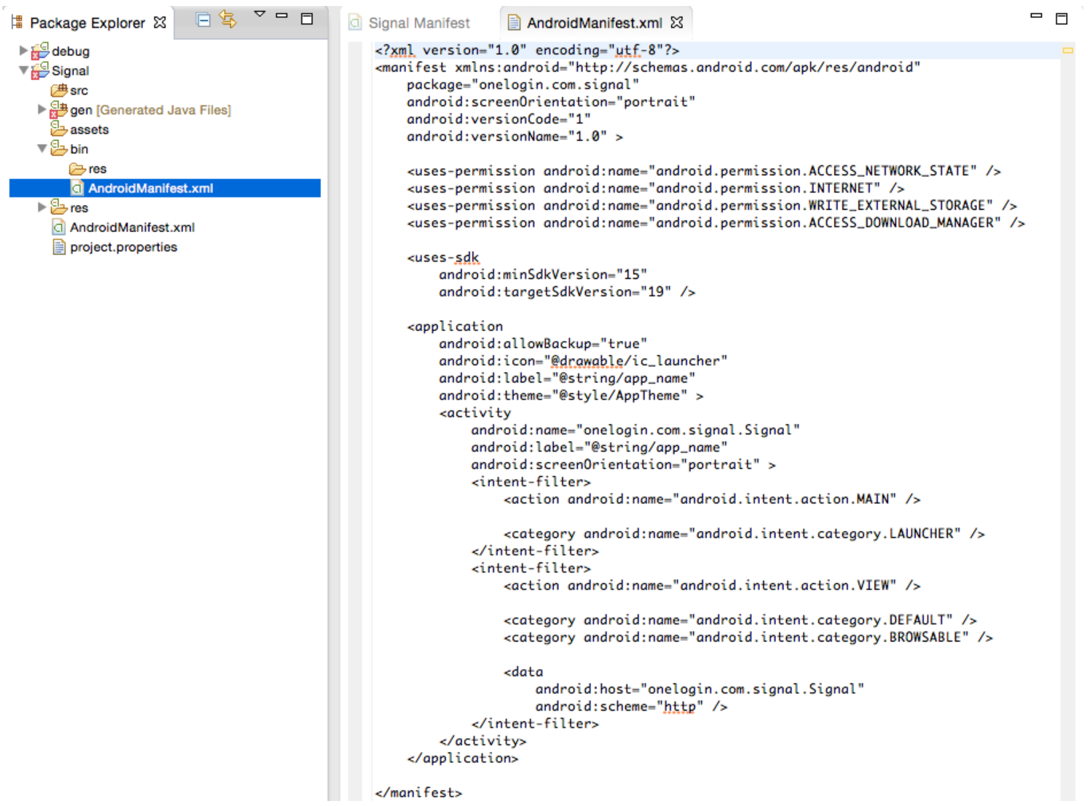


```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="onelogin.com.signal"
    android:screenOrientation="portrait">

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


    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:screenOrientation="portrait"
            android:name="onelogin.com.signal.Signal"
            android:label="@string/app_name" >

            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="android.intent.category.BROWSABLE" />
                <data android:scheme="http" android:host="onelogin.com.signal.Signal" />
            </intent-filter>

        </activity>
    </application>


</manifest>
```

To install the .jar file, navigate to **File > Project Structure**, and then after selecting **app**, select the **Dependencies** tab. Here, selecting the **+** sign at the bottom of the page will allow you to add a **File Dependency**, where you will search for and add the custom OneLogin .jar file to the SDK.

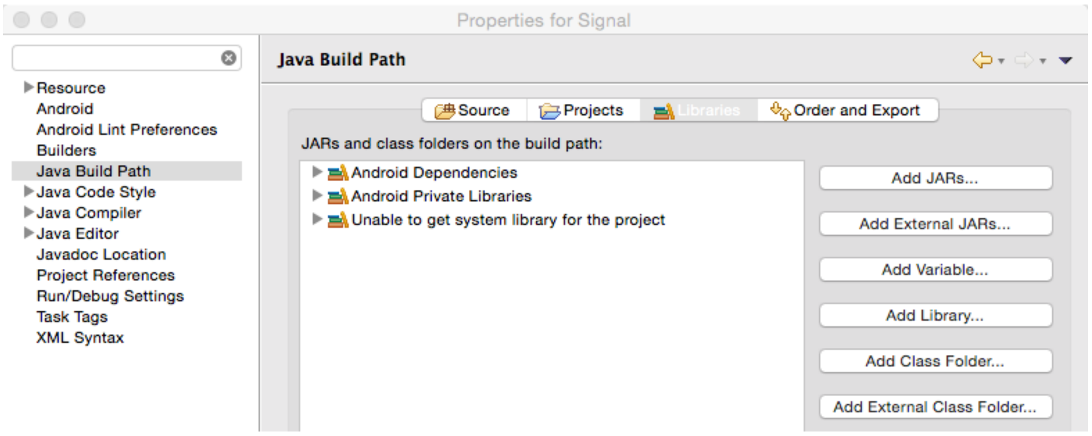
FINDING THE CUSTOM URL IN ECLIPSE

.After importing the project into Eclipse ADT, finding the required custom URL scheme is fairly straightforward. Again, you'll be looking for the **AndroidManifest.xml**, found under the directory path of **Signal > bin > AndroidManifest.xml**. Opening the file will display the following:



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="onelogin.com.signal"
    android:screenOrientation="portrait"
    android:versionCode="1"
    android:versionName="1.0" >

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

    <uses-sdk
        android:minSdkVersion="15"
        android:targetSdkVersion="19" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="onelogin.com.signal.Signal"
            android:label="@string/app_name"
            android:screenOrientation="portrait" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />

                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="android.intent.category.BROWSABLE" />

                <data
                    android:host="onelogin.com.signal.Signal"
                    android:scheme="http" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

To install the .jar file to the SDK in Eclipse, proceed to **Project > Properties** in the upper navigation bar, and then select **Java Build Path** from the left-hand list, where you'll be selecting the **Add JARs...** button.



This will then allow you to add the .jar file from its local directory.

FINDING THE CUSTOM URL IN ECLIPSE

.After importing the project into Eclipse ADT, finding the required custom URL scheme is fairly straightforward. Again, you'll be looking for the **AndroidManifest.xml**, found under the directory path of **Signal > bin > AndroidManifest.xml**. Opening the file will display the following:

To install the .jar file to the SDK in Eclipse, proceed to **Project > Properties** in the upper navigation bar, and then select **Java Build Path** from the left-hand list, where you'll be selecting the **Add JARs...** button.

This will then allow you to add the .jar file from its local directory.

RECEIVING A SECONDARY TOKEN FROM AGENT

When the token agent has successfully retrieved a secondary token or has encountered a failing error, it will use your Identifier (defined previously) as a callback to your application, passing you a token or an error.

If you receive an error, this is the list of error codes you are likely to receive:

```
OLNAPPSErrorNetworkNotAvailable                    = 1000,
OLNAPPSErrorIDPDidNotRespond                       = 1100,
OLNAPPSErrorRequestingAppDoesNotHavePermission     = 1201,
OLNAPPSErrorSecondaryTokenRequestRejected          = 1202,
OLNAPPSErrorNoSessionActive                        = 1204,
OLNAPPSErrorDeviceCouldNotBeEnrolled               = 1205,
OLNAPPSErrorTokenAgentNotInstalled                 = 1300,
OLNAPPSErrorNoSessionActive                        = 1500,
OLNAPPSErrorDeviceCouldNotBeEnrolled               = 1600,
OLNAPPSErrorNone                                   = 0
```

The errors above are described below in more detail:

Error 1000 - The network is not available
Error 1100 - There was a timeout in the request or the server is down.
Error 1201 - a. The Token Agent gets a request for a second token
b. Verify that the Requesting App is in the current App Info list.
c. If it is, a secondary token for it is acquired.
d. If it is not, the App Info list is refreshed.
e. If after refreshing, the Registering App is still not in the App Info list, return this error.
Error 1202 - If the server responds with an error after requesting an updated App Info List or a Secondary Token
Error 1203 - Couldn't retrieve the App Info List.
Error 1204 - No current active session.
Error 1205 - Device could not be enrolled.
Error 1300 - The Token Agent is not installed on the device.
Error 1500 - The Token Agent does not have the information necessary to begin a session.
Error 1600 - The device could not be enrolled with the server.
Error 0 - Everything is running successfully.

With this information, you are now ready to run the test application.

RUNNING THE TEST APPLICATION

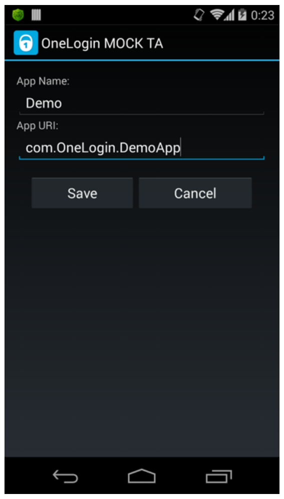
Running the test application is incredibly easy:

1. Load the demo application into your device. Right now iPhone is the current testing environment.
2. Once within the test SDK, select the toggle to enable the **Demo App**.
3. The Mock Token Agent can simulate the following error conditions.
   3.1. By selecting 'Network not available (1000)', the resultant errors become:
   - Network Not Available(1000)
   - IdP Did Not Respond(1100)

   3.2. By disabling the application in Step 2, or entering an incorrect App URL in step 4, the resultant errors become:
   - Permission Denied(1201)
4. When prompted to enter an **App Name**, enter 'demo', and for the **App URL**, enter 'DEMO'. If you wish, this can be changed on line 13 of the Mock Token Agent source code under 'URLscheme ='.
5. With everything complete, the test application should run successfully and prompt you with a response saying that the secondary tokens was successfully generated by the Token Agent.

   When in Production, you will have your secondary token in your application to be used against a secondary resource server.
6. And with that you're done!

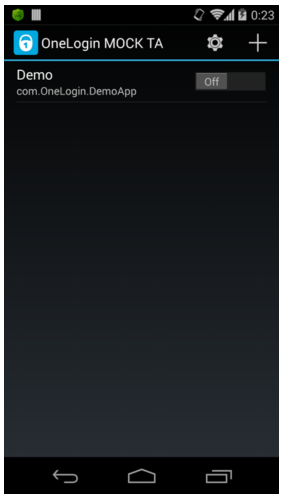
2.


OneLogin MOCK TA

App Name: Demo

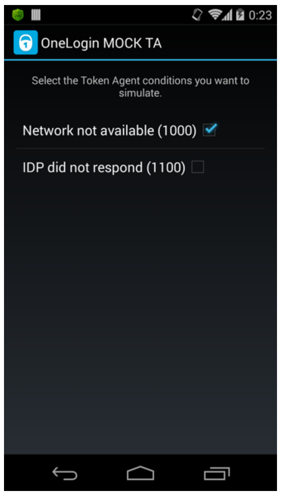
App URI: com.OneLogin.DemoApp

3.


OneLogin MOCK TA

Demo
com.OneLogin.DemoApp
Off

4.


OneLogin MOCK TA

Select the Token Agent conditions you want to simulate.

Network not available (1000) ✓

IDP did not respond (1100)

The Test Application Source Code

```
<intent-filter>
            <action android:name="android.intent.action.MAIN" />

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

         <intent-filter>
            <action android:name="android.intent.action.VIEW" />
            <category android:name="android.intent.category.DEFAULT" />
            <category android:name="android.intent.category.BROWSABLE" />
            <data android:scheme="http" android:host="onelogin.com.signal.Signal" />
         </intent-filter>


public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      //init lib
      OLNAPPS = OneLogin.SDK(this);

      Uri data = getIntent().getData();

         try {
            isTAInstaled = OLNAPPS.isTokenAgentInstalled();
         } catch (Exception ex) {
            ex.printStackTrace();
            Log.e(TAG, ex.getMessage());
         }
         //if TA is installed, start TA to get secondary token

                                 if (isTAInstaled) {
            //start TA
            OLNAPPS.requestTokenForAppWithURLScheme("http://onelogin.com.signal.Signal://");
         }

   }
```

Test Application Endpoint for Secondary Token from

```
Uri data = getIntent().getData();
      //get result
      if (data != null) {
         //get secondary token
         Token token = OLNAPPS.getToken(data);
         if (token.getError() == 0) {
            Toast.makeText(this, "Token: " + token.getSecondaryToken(), Toast.LENGTH_LONG).show();
            //continue work program
            initList();
         } else {
            Toast.makeText(this, "Error: " + token.getError(), Toast.LENGTH_LONG).show();
         }
      }
```