Quick Start Guide for Android APS SDK

Created by Brian Immel, last modified on Apr 23, 2019

This AMPLIFY Appcelerator Services feature requires a Pro or Enterprise Subscription.

Introduction

This guide walks through the setup of the AMPLIFY Appcelerator Services for Android applications. The AMPLIFY Appcelerator Services SDK gives you access to the Appcelerator Analytics, Cloud, and Performance services.

We also provide service and features for non-Android platforms in our AMPLIFY Appcelerator Services documentation.

Requirements

This document assumes you have an existing Android application and are familiar with the Android toolchain to create, build, and package Android applications. The following prerequisites are also needed:

- Android SDK 2.3.x (API Level 10) or greater **(used API 22 with Android Studio)**
- Android NDK
- Android Studio (this guide uses version 3.1.2 or later)

Setup

This section provides information on the prerequisites to useAppcelerator Services in your application.

Before you can use Appcelerator Services in your application, you need to:

- Create an application in the Dashboard (see Dashboard Basics for more information.)
- Get the application keys by reviewing Setting Up Application Keys.

Register an APS SDK application

Appcelerator Platform Services (APS) SDK for iOS and Android provides support for Appcelerator Analytics, Cloud, and Performance services for your Android applications built with the native Android APIs and Java and iOS applications built with the native iOS APIs and Objective-C or Swift.

To register an APS SDK application for services:

1. Sign into the AMPLIFY Platform.
2. Click the **Add menu** (+) and select **Register App for Services** to open the *Register App* for Services form.
3. Enter the **Name** of the application.
4. Select **APS SDK** from the *Type* selection menu.
5. Select a **Platform** (Andriod or iOS).
6. Optional (but recommended), enter a unique **Identifier** for your application.
7. Optional (but recommended), enter a **Description** for your application.
8. Select **Services** for your application by selecting or deselecting the checkboxes for the following:
    - Analytics
    - Provision Cloud Services (Mobile Backend Services)
    - Performance
9. **Add application team members** from your organization by clicking the **add (+)** button in the App Team list.
10. Click **OK** to finish registering your application.

Appcelerator Dashboard displays the Services tab for your application. Follow the directions to add Platform Services to your application.

Register an API or microservice application

To register an API or microservice application:

1. Sign into the AMPLIFY Platform.
2. Click the Add menu (+) and select Register App for Services to open the Register App for Services form.
3. Enter the **Name** of the application.
4. Select **API/Microservice** from the *Type* selection menu.
5. Enter a **Platform** for your application.
6. Optional (but recommended), enter a unique **Identifier** for your application.
7. Optional (but recommended), enter a **Description** for your application.
8. Select **Services** for your application by selecting or deselecting the checkboxes for the following:
    - Analytics
    - Provision Cloud Services (Mobile Backend Services)
    - Performance
9. **Add application team members** from your organization by clicking the **add (+)** button in the App Team list.
10. Click **OK** to finish registering your API or microservice.

Register a website or web application

To register a Website or Web application:

1. Sign into the AMPLIFY Platform.
2. Click the Add menu (+) and select Register App for Services to open the Register App for Services form.
3. Enter the **Name** of the application.
4. Select **Website/Web App** from the *Type* selection menu.
5. Enter a **Platform** for your application.

6. Optional (but recommended), enter a unique **Identifier** for your application.
7. Optional (but recommended), enter a **Description** for your application.
8. Select **Services** for your application by selecting or deselecting the checkboxes for the following:
    - Analytics
    - Provision Cloud Services (Mobile Backend Services)
    - Performance
9. **Add application team members** from your organization by clicking the **add (+)** button in the App Team list.
10. Click **OK** to finish registering your website or web application.

Register a Custom application

To register a custom application (other than APS SDK, API/Microservice, or Website/Web applications):

1. Sign into the AMPLIFY Platform.
2. Click the Add menu (+) and select Register App for Services to open the Register App for Services form.
3. Enter the **Name** of the application.
4. Select **Other** from the *Type* selection menu.
5. Enter the **Name** of the application.
6. Optional (but recommended), enter a unique **Identifier** for your application.
7. Optional (but recommended), enter a **Description** for your application.
8. Select **Services** for your application by selecting or deselecting the checkboxes for the following:
    - Analytics
    - Provision Cloud Services (Mobile Backend Services)
    - Performance
9. **Add application team members** from your organization by clicking the **add (+)** button in the App Team list.
10. Click **OK** to finish registering your custom application.

From here, you will need to download the APS SDK from either this link or by navigating via the Dashboard (Cloud/Performance/Analytics tab should be open by default).

For more information, refer to Managing Non-Titanium Client Applications in the Dashboard.

Basic APS tutorial

The following tutorial demonstrates basic setup and usage of Analytics, Cloud, and Performance libraries in an Eclipse project. To complete the tutorial, you will need your application key for the Cloud, Analytics, and Performance services. You can skip ahead and download a complete version of the project.

To create a basic application using APS:

1. In **Android Studio**, go to **File** > **New** > **New Project....**
2. Enter **<name of app>** in the **Application name** field.
3. Select **Phone and Tablet** from the list of options in the **Select form factors and minimum SDK** fields.
4. Select **API 22: Android 5.1 (Lollipop)** and click on **Next**. Note: this API version was choosen at the time of writing this document. This tutorial was tested against this version and may not work with other versions.
5. In **Add an Activity to Mobile**, select **Empty Activity** and click on **Next**.
6. In the **Configure Activity**, leave the default names as is and click on **Next**. The Component Installer will start installing the requested components based on these selections. It may take a minute or two to complete this request. Once it's finish, click the **Finish** button.
7. **Unpack the appcelerator-sdk-android-<VERSION>.zip** file.
8. **Copy the appcelerator-sdk-android-<VERSION>.jar** file to the `/libs` folder of your Android project (if you wish to use Maven or Gradle, see "Building the application" below for setup instructions).
9. Modify the project's app/manifests/AndroidManifest.xml file to include the ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE, REAL_GET_TASKS, INTERNET, and READ_LOGS and WRITER_EXTERNAL_STORAGE user permissions and declare the com.appcelerator.aps.AnalyticsService

as a Service class, which allows the APS library to send analytic events to the APS servers while the application is in the background:
AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.appcelerator.sample">
     <!-- Add these permissions to enable Analytics, Cloud and Performance -->
     <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
     <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
     <uses-permission android:name="android.permission.INTERNET"/>
     <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
     <uses-permission android:name="android.permission.REAL_GET_TASKS" />
     <uses-permission android:name="android.permission.READ_LOGS"/>
     <application>
          <activity>
               ...
          </activity>

          <!-- Add this service to enable Analytics -->
          <service android:name="com.appcelerator.aps.APSAnalyticsService"
android:exported="false" />
     </application>
     </manifest>
```

11. Add the following import statement to the project:
MainActivity.java

```
...
import com.appcelerator.aps.APSServiceManager;
```

13. In the MainActivity's `onCreate()` method, add the following method call to enable the APS services.
MainActivity.java

```
APSServiceManager.getInstance().enable(getApplicationContext(), "APP_KEY");
```

The Android application is now ready to make method calls using the APS SDK APIs.

Modify the application

Customize the application's UI to display a spinner, text field and button, and add some logic to respond to user interaction. The spinner will display a list of available user accounts. The user can enter their password in the text field, then click the button to log in.

1. Open the fragment layout XML file (`res/layout/fragment_main.xml` in the Graphical Layout editor.
2. Remove the "Hello World!" label.
3. Drag a Spinner widget, EditText widget (password text field) and Button widget into the view.
4. In the `MainActivity.java` file, modify the code to save an instance of the current activity, Spinner and EditText widgets. Modify the application to bind a `doClick` method to the Button's `onClick` listener and create an empty function called `populateSpinner`. You need to also import additional packages. In the following sections, you will add code to these handlers that call the Cloud, Analytics, and Performance services.

MainActivity.java

```
// Import the following packages
import android.app.Activity;
import android.util.Log;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;
import java.util.HashMap;
import org.json.JSONArray;
import com.appcelerator.aps.APSAnalytics;
import com.appcelerator.aps.APSCloudException;
import com.appcelerator.aps.APSPerformance;
import com.appcelerator.aps.APSResponse;
import com.appcelerator.aps.APSResponseHandler;
import com.appcelerator.aps.APSServiceManager;
import com.appcelerator.aps.APSUsers;

public class MainActivity extends ActionBarActivity {
     // Handle to current activity
     private static Activity currentActivity;

     @Override
     protected void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.activity_main);

          // Save the current activity
          currentActivity = this;

          APSServiceManager.getInstance().enable(getApplicationContext(), "APP_KEY");

          if (savedInstanceState == null) {
               getSupportFragmentManager().beginTransaction()
               .add(R.id.container, new PlaceholderFragment())
               .commit();
          }
     }

     public static class PlaceholderFragment extends Fragment {
          // Handle to Spinner and EditText widgets
          Spinner spinner;
          EditText textField;

          public PlaceholderFragment() {
          }

          @Override
          public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
               View rootView = inflater.inflate(R.layout.fragment_main, container, false);

               // Bind the Button do the doClick method
               Button button = (Button)rootView.findViewById(R.id.button1);
               button.setOnClickListener(new Button.OnClickListener() {
                    @Override
                    public void onClick(View v){
                         doClick();
                    }
```

```
            }
            // Save the Spinner and EditText instances
            spinner = (Spinner)rootView.findViewById(R.id.spinner1);
            textField = (EditText)rootView.findViewById(R.id.editText1);
            // Place holder for the next tutorial steps
            populateSpinner();
            return rootView;
        }
        // Placeholder for next tutorial steps
        public void doClick() {
        }

        public void populateSpinner() {
        }
    }
 }
```

Building the application

If you use Maven or Gradle as part of your build process, you will need to publish the APS SDK to your local Maven repository and add the APS SDK as a dependency to your project.

First, unzip the APS SDK and install the dependencies into your local Maven repository:

```
unzip appcelerator-sdk-android-1.0.0.zip
mvn install:install-file -Dfile=appcelerator-sdk-android-1.0.0/appcelerator-sdk-android-1.0.0.jar -
DgroupId=com.appcelerator -DartifactId=appcelerator-sdk-android -Dversion=1.0.0 -Dpackaging=jar
```

Next, add the following lines of code to your Maven or Gradle POMs:

Gradle

```
dependencies {
     compile 'com.appcelerator: appcelerator-sdk-android:1.0.0'
}
```

Maven

```
<dependencies>
     <dependency>
          <groupId>com.appcelerator</groupId>
          <artifactId>appcelerator-sdk-android</artifactId>
          <version>1.0.0</version>
     </dependency>
</dependencies>
```

Capture user session events

Unlike the iOS and Titanium platforms, you need to explicitly make analytic method calls in the application to send user session events to the Analytics service to capture user activity.

Use the Activity's lifetime events to track when the user is actively using your application. Call the APSAnalytics' `sendAppEnrollEvent()` method in the `onCreateMethod()`, which indicates the first installation or upgrade of the application.

Before calling API calls to the APSAnalytics class, you need to get a shared instance of the APSAnalytics class using the `getInstance()` method.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);
     APSAnalytics.getInstance().sendAppEnrollEvent();

     // Other init calls...
}
```

Use `onPause()` and `onResume()` to call the APSAnalytics' `sendAppBackgroundEvent()` and `sendAppForegroundEvent()`, respectively. These two APSAnalytics methods send user session events to the APS analytics servers to track how long a user engages with your application.

```
@Override
public void onPause() {
     super.onPause();
     APSAnalytics.getInstance().sendAppBackgroundEvent();
}

@Override
public void onResume() {
     super.onResume();
     APSAnalytics.getInstance().sendAppForegroundEvent();
}
```

Send an Analytics feature event

Besides user session events, you can also send custom analytics events, as shown in this example. There are two types of custom events: feature events and navigation events.

In the `doClick()` function, add a APSAnalytics' `sendAppFeatureEvent()` method call to send a feature event with the string `"sample.feature.login"`. The optional second parameter is set to null for this example, but you can send additional data as a JSON object with the event.

MainActivity.java

```
public void doClick(View view) {
     APSAnalytics.getInstance().sendAppFeatureEvent("sample.feature.login", null);
};
```

Query Cloud users

To use the APS Cloud component, most of the methods require a user to be logged in. This sample uses the Spinner widget to select a Cloud user accounts. To populate the Spinner values, the application needs to retrieve a list of users. Use the `APSUsers.query()` method to retrieve a list of user accounts.

Every APS Cloud method includes a handler parameter that specifies the callback to handle the server response. The callback is passed an APSResponse object that contains response metadata (such as success or failure) and the response payload.

MainActivity.java

```
public void populateSpinner() {
     try {
          APSUsers.query(null, new APSResponseHandler() {
               @Override
               public void onResponse(final APSResponse e) {
                    if (e.getSuccess()) {
                         try {
                              JSONArray payload = e.getResponse().getJSONArray("users");
                              String[] items = new String[payload.length()];
                              for (int i = 0; i < payload.length(); i++) {
                                   items[i] = payload.getJSONObject(i).getString("username");
                              }
                              ArrayAdapter spinnerArrayAdapter = new ArrayAdapter(currentActivity,
                                   android.R.layout.simple_spinner_item, items);
                                   spinner.setAdapter(spinnerArrayAdapter);
                              } catch (Exception ex) {
                                   Log.e("ACSUsers", "Error parsing JSON object: " +
ex.toString());
                              }
                         } else {
                              Toast.makeText(currentActivity, "ERROR: Unable to get users.",
Toast.LENGTH_SHORT).show();
                              Log.e("ACSUsers", e.getResponseString());
                         }
                    }

                    @Override
                    public void onException(APSCloudException e) {
                         Log.e("ACSUsers", e.toString());
                    }
               });
          } catch (APSCloudException e) {
               Log.e("ACSUsers", e.toString());
          }
     }
```

Login to a Cloud account

To login to a Cloud account, you need the username and password. Since the application was modified to get all available user accounts and populate the Spinner, the application needs to get the current value of the spinner and the text entered in the EditText widget. These values are passed to the `APSUsers.login()` method. Modify the `doClick()` method to login to a Cloud user account.

```
public void doClick() {
     APSAnalytics.getInstance().sendAppFeatureEvent("sample.feature.login", null);

     // Get the current value of the Spinner and EditField widgets
     final String username = spinner.getSelectedItem().toString();
     String password = textField.getText().toString();

     // Use a HashMap to send the method parameters for the request
     HashMap<String, Object> data = new HashMap<String, Object>();
     data.put("login", username);
     data.put("password", password);

     try {
          APSUsers.login(data, new APSResponseHandler() {
               @Override
               public void onResponse(final APSResponse e) {
                    if (e.getSuccess()) {
                         Log.i("ACSUsers", "Successfully logged in as " + username);
                    }
                    else {
                         Log.e("ACSUsers", e.getMessage());
                    }
               }

               @Override
               public void onException(APSCloudException e) {
                    Log.e("ACSUsers", e.toString());
               }
          }
     } catch (APSCloudException e) {
          Log.e("ACSUsers", e.toString());
     }
}
```

Log a handled exception

The Performance library automatically logs application crashes (unhandled exceptions) and handled exceptions to the backend Performance service. You can also leave breadcrumbs in your application, which are developer-defined text strings (up to 140 characters) that are analogous to log messages.

For example, you can replace the `Log` calls in the catch statements with the APSPerformance's `logHandledException()` calls. Instead, the application will generate a runtime exception, and then call the `logHandledException()` method to log that exception to the Performance backend.

Before making API calls to the APSPerformance class, you need to retrieve a shared instance of the class using the `getInstance()` method.

To the `doClick` method, add the following new code:

MainActivity.java

```
public void doClick() {
     // Analytics call...
     // Cloud call...
     try {
          throw new Exception("Something happened...");
     } catch (Exception exception) {
          APSPerformance.getInstance().logHandledException(exception);
     }
}
```

Set a username for crash logs

To help differentiate crash logs, use the APSPerformance's `setUsername()` method. When the application successfully logs in to the Cloud user account, the application calls APSPerformance's `setUsername()`method.

```
APSUsers.login(data, new APSResponseHandler() {
     @Override
     public void onResponse(final APSResponse e) {
          if (e.getSuccess()) {
               Log.i("ACSUsers", "Successfully logged in as " + username);

               // Add this method call
               APSPerformance.getInstance().setUsername("username");
               } else {
                    Log.e("ACSUsers", e.getMessage());
               }
     }
});
```

Testing the tutorial sample

Before testing the sample, you need to create user accounts for the application to query. To create Cloud user accounts:

1. Log in to **Appcelerator Dashboard**.
2. In Dashboard, select your application from the **APIs menu** in the upper-left corner.
3. In the left navigation bar, click **Manage Data**.
4. Click **Users**, then the **Create User** button.
5. In the **Username** field enter the user's username.
6. In the **Password** field enter the new user's password (four-character minimum).
7. Click Save **Changes**.

To create a Cloud user account, you only need a username or e-mail address, and a password.

For more information about managing Cloud user accounts, see Managing Organizations.

After you have created a few Cloud user accounts, build the sample and launch it on either a device or emulator. Once the application launches:

1. Click on the **Picker/Spinner**. You should see a list of all the Cloud user accounts you added.
2. Select a **user account** from the Picker/Spinner and enter that user's password. Click the **Next** button. In the log output, you should see an info log that login command was successful or not.
3. Go to the **Appcelerator Dashboard**.
4. In Dashboard, select your application from the **Apps** menu in the upper-left corner.

5. In the left navbar, click **Performance**, then **Handled Exceptions**. You should see the "Something happened..." exception in the list.
6. In the left navbar, click **Search by User**. Enter the username of the account that successfully logged in. Click the username. You should see a crash report for the user.
7. In the left navbar, select **Analytics**.
8. In the **Real Time** view, you should see at least one active session.
9. In the left navbar, click **Events**. You should see the `"sample.feature.login"` feature event.

Next Steps for Appcelerator Analytics, Cloud and Performance

This tutorial only covers a small portion of what the Analytics, Cloud, and Performance services can provide. For more in-depth information about these features, see the following topics:

- APS Analytics for Android
- AMPLIFY Appcelerator Platform Services SDK for Android Mobile Backend Services
- APS Performance for Android