Adding the MobileFirst Platform Foundation SDK to Android Applications

Overview

Unlike MobileFirst Hybrid applications, which get autogenerated with the MobileFirst JavaScript SDK built-in and are thus capable to communicate with the MobileFirst Server out-of-the-box, native apps are not bundled with the MobileFirst Native SDK by default.

In this tutorial, you learn how to add the MobileFirst Native SDK to a native Android application and set it up with additional required configuration files.

By using MobileFirst Native SDK, you can implement various features, such as: connecting to the MobileFirst Server instance, calling adapter procedures, implementing authentication methods, certificate pinning, using push notifications, and more.

For a request from a native app to be recognized by MobileFirst Server, the client application must be registered to the MobileFirst Server instance.

In this tutorial, you learn how to deploy the server-side entity for the native Android application.

Prerequisite

- Proficiency with the Google Android Studio
- If you haven't gone through the Command Line Interface tutorial (/../../advanced-client-side-development/updated-using-cli-to-create-build-and-manage-mobilefirst-project-artifacts/), do so now.

Agenda

- Adding the MobileFirst Native SDK to an application
- Deploying the server-side entity to MobileFirst Server
- Inside a native application containing the MobileFirst Native SDK
- Tutorials to follow next

Adding the MobileFirst Native SDK to an application

The MobileFirst Native SDK can be added in several ways: from a remote MavenCentral repository, ensuring you always use the latest available version, or locally by using either MobileFirst Studio plug-in for Eclipse or the MobileFirst CLI tool. The preferred method is by using Jcenter, explained next. For the "Classic" local method, see the local method section.

1. Create an Android Studio project or use an existing one.
2. In **Project > Gradle scripts**, select **build.gradle (Module: app)**.
3. Add the following lines below `apply plugin: 'com.android.application'`:
   [code lang="xml"]repositories{
   jcenter()
   }[/code]
4. Add the following inside `android`:
   [code lang="xml"]packagingOptions {

pickFirst 'META-INF/ASL2.0'
pickFirst 'META-INF/LICENSE'
pickFirst 'META-INF/NOTICE'
}[/code]

5. Add the following lines inside `dependencies`:
   [code lang="xml"]compile group: 'com.ibm.mobile.foundation',
   name: 'ibmmobilefirstplatformfoundation',
   version: '7.1.0.0',
   ext: 'aar',
   transitive: true[/code]
6. Add the following permissions to the `AndroidManifest.xml` file:
   [code lang="xml"]
   <uses-permission android:name="android.permission.INTERNET"/>
   <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
   <uses-permission android:name="android.permission.GET_TASKS" />
   [/code]
7. Add the MobileFirst UI activity:
   [code lang="xml"]<activity android:name="com.worklight.wlclient.ui.UIActivity" />[/code]

Deploying the server-side entity to MobileFirst Server

Now that the native application is set up with the MobileFirst Native SDK, it is also required that MobileFirst Server recognize it after the SDK is used in the application. For this, you need a MobileFirst Server instance and a MobileFirst project:

Creating a project and server instance

In **Terminal**, navigate to the location where you want to create the MobileFirst Backend project and run the following commands:

[code lang="xml"]mfp create your-project-name-here
cd your-project-name
mfp start[/code]

After the project is created and the server is running, the server-side entity can be created and deployed to the server.
The next step adds the following files to the native project, as explained in Inside a native application containing the MobileFirst Native SDK: `wlclient.properties`, `application-descriptor.xml`, `mobilefirst` folder.

Deploying the server-side entity

1. In **Terminal**, navigate to the root of the Android Studio project.
2. Run the command: `mfp push`
   The command asks to which MobileFirst Server you want to deploy the artifacts and update the `wlclient.properties` file accordingly with the server address. You can later edit these settings again manually if necessary. You can also edit the files by using the CLI command `mfp config`.

Inside a native application containing the MobileFirst Native SDK

In addition to the reconfigured project, now containing the MobileFirst Native SDK, three artifacts were added:

wlclient.properties

The `wlclient.properties` file, located in the `android-studio-project\app\src\main` folder, holds server configuration properties and is user-editable:

- `wlServerProtocol` – The communication protocol to MobileFirst Server, which can be either `http` or `https`.
- `wlServerHost` – The hostname of MobileFirst Server.
- `wlServerPort` – The port of MobileFirst Server.
- `wlServerContext` – The context root path of the application on MobileFirst Server.
- `wlAppId` – The application ID as defined in the `application-descriptor.xml` file.
- `wlAppVersion` – The application version.
- `wlEnvironment` – The target environment of the native application.
- `wlUid` – The property used by MTWW, the test workbench, to identify this as a MobileFirst application.
- `wlPlatformVersion` – The MobileFirst SDK version.
- `languagePreferences` – The list of preferred locales.
- `GcmSenderId` – This property defines the Google Cloud Message (GCM) Sender ID to be used for push notifications. By default, this property is commented out.

Application descriptor

The `application-descriptor.xml` file, located at the root of the project, is a metadata file that is used to define various aspects of the application, such as user identity realms and push notifications support, security settings that MobileFirst Server enforces, and more.

`mobilefirst` folder

The `mobilefirst` folder, located at the root of the project contains .wlapp files. These are the server-side entities that are deployed to the MobileFirst Server.

Tutorials to follow next

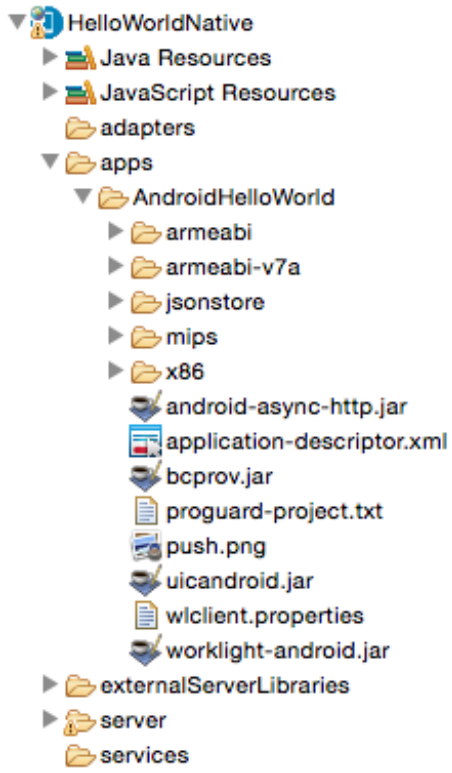
Now that your application contains the Native API library, you can follow the tutorials in the Native Android Development (../../native/android) section to learn more about authentication and security, server-side development, advanced client-side development, notifications and more.

"Classic" local method to add the MobileFirst Native SDK to an application

When using the local method, this means that the SDK files will be generated from the existing local resources in the installed version of MobileFirst Platform Foundation. In the "Classic" method, the SDK files are referred to as "NativeAPI". Follow the instructions below for either the MobileFirst Studio plug-in for

Eclipse or MobileFirst CLI.

Using either Studio or CLI, create and deploy the NativeAPI server-side entity:



(http://developer.ibm.com/mobilefirstplatform/wp-content/uploads/sites/32/2014/12/02_03_android_native_folder_structure-7-0.png)

1. # Studio

    1. In MobileFirst Studio, create a MobileFirst project and add a MobileFirst Native API.
    2. In the **New MobileFirst Native API** dialog, enter your application name and select **Android** for the **Environment** field.
    3. Right-click the generated NativeAPI folder (located in *`your-projects`*`/apps/`*`your-nativeapi-app-name`*) and select **Run As > Deploy Native API**.
       Note: This action is required for MobileFirst Server to recognize the application when a request reaches the server.

2. # CLI

    1. In the terminal with the CLI (../../updated-using-cli-to-create-build-and-manage-mobilefirst-project-artifacts/) installed, create a new MobileFirst project: `$ mfp create HelloWorldNative`
    2. Go to the newly created project directory: `$ cd HelloWorldNative/`
    3. Add a new Android native API: `$ mfp add api AndroidHelloWorld -e android`
    4. navigate to the Android project and run the command: `$ mfp push`.
       This action is required for MobileFirst Server to recognize the application if it attempts to connect.

3. Copy the `worklight-android.jar`, `uicandroid.jar`, `bcprov.jar`, and `android-async-`

`http.jar` files from the generated NativeAPI folder to the new native Android application, in the `/libs` directory.

4. [Android Studio only] Right-click any of the added `.jar` files and select **Add as library** to add all libraries.
5. [Android Studio only] Create an `assets` folder under `src\main`.
6. Copy the `wlclient.properties` file from the previously created MobileFirst native API folder to the new native Android application, in the `/assets` directory.
7. Add the following permissions to the `AndroidManifest.xml` file:
   [code lang="xml"]
   <uses-permission android:name="android.permission.INTERNET"/>
   <uses-permission android:name="android.permission.ACCESS*WIFI*STATE"/>
   <uses-permission android:name="android.permission.GET_TASKS" />
   [/code]
8. Add the MobileFirst UI activity:
   [code lang="xml"]<activity android:name="com.worklight.wlclient.ui.UIActivity" />[/code]

The NativeAPI contains the following components:

- The `wlclient.properties` file contains the connectivity settings that a native Android application uses.
- The `worklight-android.jar` file is the MobileFirst API library.
- The `push.png` file is used for displaying an image for an incoming push notification.
- The `proguard-project.txt` file contains configuration settings for using Android ProGuard with Android Native API.
- The `JSONStore` folder is for optional JSONStore support in native applications.
- The `armabi`, `armabi-v7a`, `mips`, and `x86` folders are for application authenticity protection (optional) in native applications.
- As with any MobileFirst project, you create the server configuration by modifying the files that are in the `server\conf` folder.
- You use the `application-descriptor.xml` file to define application metadata and to configure security settings that MobileFirst Server enforces.

> For more information, see the topic about developing native applications for Android, in the user documentation.