Handling Push Notifications in Android applications

fork and edit tutorial (https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/8.0/notifications/handling-push-notifications-in-android.md) | report issue (https://github.ibm.com/MFPSamples/DevCenter/issues/new)

The handling in client side tutorials should explain: - how to setup push notifications support in iOS Xcode project (editing the podfile?) - how to setup push notifications support in Andrid Studio project (editing the builde.gradle file?) - how to setup push notifications support in Cordova applications - how to intercept and display notifications in the client

Overview

Before Android applications are able to handle any recieved push notifications, they must configured with support for Google Play Services. Once an application has been configured, MobileFirst-provided Notifications API can be used in order to register & unregister devices, and subscribe & unsubscribe to tags.

In this tutorial you learn how to configure an Android application and how to use the MobileFirst-provided Notifications API.

Prerequisites:

- Android Studio and MobileFirst Developer CLI installed on the developer workstation.
- MobileFirst Server to run locally, or a remotely running MobileFirst Server.
- Make sure you have read the Setting up your MobileFirst development environment (../../../setting-up-your-development-environment/index) tutorial, as well as the Adding the MobileFirst Platform Foundation SDK to Android applications (../../../adding-the-mfpf-sdk/android) tutorial.

Jump to:

Notifications Configuration

Create a new Android Studio project or use an existing one.
If the MobileFirst Native Android SDK is not already present in the project, follow the instructions in the Adding the MobileFirst Platform Foundation SDK to Android applications (../../../adding-the-mfpf-sdk/android) tutorial.

Project setup

1. In **Android → Gradle scripts**, select the **build.gradle (Project: [application-name])** file.
    1. Add the following line to `dependencies`:

    ```
    classpath 'com.google.gms:google-services:2.0.0-alpha3'
    ```

2. In **Android → Gradle scripts**, select the **build.gradle (Module: app)** file.
    1. Add the following line to `dependencies`:

```
com.google.android.gms:play-services-gcm:8.4.0
com.squareup.okhttp:okhttp:2.6.0
```

2. Add the following line at the bottom:

```
apply plugin: 'com.google.gms.google-services'
```

Note: This line must be placed at the bottom of `app/build.gradle` as to not create an dependency collisions. For more details see *The Google Services Gradle Plugin* (https://developers.google.com/android/guides/google-services-plugin)

remove step 3 before going live

3. Copy ibmmobilefirstplatformfoundationpush-1.0.0.aar (from halpert Electra DevOps Latest integration build) to `<android_sdk>\extras\google\m2repository\com\ibm\mobile\foundation\ibmmobilefirstplatformfoundationpush\1.0.0\ibmmobilefirstplatformfoundationpush-1.0.0.aar`

   Remove libs folder from the aar.

   Note: This step is not required once the lib gets to maven central/jcenter. Just need to add mavenCentral()/jcenter() in app gradle.

4. Add the push required configuration in AndroidManifest.xml
   1. Add the following permissions to the top the `manifest` tag:

```
<!-- Permissions -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.GET_TASKS" />
<uses-permission android:name="android.permission.WAKE_LOCK" />

<!-- GCM Permissions -->
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
<permission
    android:name="your.application.package.name.permission.C2D_MESSAGE"
    android:protectionLevel="signature" />
```

   2. Add the following `MobileFirst UI Activity`, `MFPPush Intent Service`, `MFPPush Instance ID Listener Service` to the `application` tag:

```
<!-- MobileFirst UI Activity -->
<activity android:name="com.worklight.wlclient.ui.UIActivity" />

<!-- GCM Receiver -->
<receiver
   android:name="com.google.android.gms.gcm.GcmReceiver"
   android:exported="true"
   android:permission="com.google.android.c2dm.permission.SEND">
   <intent-filter>
      <action android:name="com.google.android.c2dm.intent.RECEIVE" />
      <category android:name="your.application.package.name" />
   </intent-filter>
</receiver>

<!-- MFPPush Intent Service -->
<service
   android:name="com.ibm.mobilefirstplatform.clientsdk.android.push.api.MFPPushIntentServi
ce"
   android:exported="false">
   <intent-filter>
      <action android:name="com.google.android.c2dm.intent.RECEIVE" />
   </intent-filter>
</service>

<!-- MFPPush Instance ID Listener Service -->
<service
   android:name="com.ibm.mobilefirstplatform.clientsdk.android.push.api.MFPPushInstanceID
ListenerService"
   android:exported="false">
   <intent-filter>
      <action android:name="com.google.android.gms.iid.InstanceID" />
   </intent-filter>
</service>
```

Note: Be sure to replace `your.application.package.name` with the actual package name of your application.

Google Services setup

Idan: I would consider moving this entire section to the server-side setup in the overview.

To setup the Android project with Google Services, visit Google's Services website (https://developers.google.com/mobile/add?platform=android&cntapi=gcm&cnturl=https:%2F%2Fdevelopers.google.com%2Fcloud-messaging%2Fandroid%2Fclient&cntlbl=Continue%20Adding%20GCM%20Support&%3Fconfigured%3Dtrue).

1. Provide your application name and package name.
2. Select "Cloud Messaging" and click on **Enable Google cloud messaging**.

This step generates a `Server API Key` and a `Sender ID`.
The generated values are used to identify the application by Google's GCM service in order to send notifications to the device.

TODO: Add explanation what to do with these values (= add them in the Console).

Notifications API

TODO: Add introduction text to the API.

API methods for tag notifications

- `WLPush.subscribeTag(tagName,options)` - Subscribes the device to the specified tag name.
- `WLPush.unsubscribeTag(tagName,options)` - Unsubscribes the device from the specified tag name
- `WLPush.isTagSubscribed(tagName)` - Returns whether the device is subscribed to a specified tag name

API methods for tag and broadcast notifications

- `WLNotificationListener` - Defines the callback method to be notified when the notification arrives.
- `client.getPush().setWLNotificationListener(listener)` - This method sets the implementation class of the `WLNotificationListener` interface.
- `client.getPush().setOnReadyToSubscribeListener(listener)` - This method registers a listener to be used for push notifications. This listener should implement the `onReadyToSubscribe()` method.
- The `onMessage(props,payload)` method of `WLNotificationListener` is called when a push notification is received by the device. --* *props* – A JSON block that contains the notifications properties of the platform. --* *payload* – A JSON block that contains other data that is sent from MobileFirst Server. The JSON block also contains the tag name for tag-based or broadcast notification. The tag name appears in the “tag” element. For broadcast notification, the default tag name is `Push.ALL`.

Handling a push notification

Handling a secure push notification