Handling Push Notifications in Android

Overview

Before Android applications are able to handle any received push notifications, support for Google Play Services needs to be configured. 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 will learn how to handle push notification in Android applications.

Prerequisites:

- Make sure you have read the following tutorials:
  - Setting up your MobileFirst development environment (../../setting-up-your-development-environment/)
  - Adding the MobileFirst Foundation SDK to Android applications (../../adding-the-mfpf-sdk/android)
  - Push Notifications Overview (../)
- MobileFirst Server to run locally, or a remotely running MobileFirst Server.
- MobileFirst CLI installed on the developer workstation

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 Foundation SDK to Android applications (../../adding-the-mfpf-sdk/android) tutorial.

Project setup

1. In **Android → Gradle scripts**, select the **build.gradle (Module: app)** file and add the following lines to `dependencies`:

   ```
   com.google.android.gms:play-services-gcm:9.0.2
   ```

   - **Note:** there is a known Google defect (https://code.google.com/p/android/issues/detail?id=212879) preventing use of the latest Play Services version (currently at 9.2.0). Use a lower version.

   And:

```
compile group: 'com.ibm.mobile.foundation',
      name: 'ibmmobilefirstplatformfoundationpush',
      version: '8.0.+',
      ext: 'aar',
      transitive: true
```

Or in a single line:

```
compile 'com.ibm.mobile.foundation:ibmmobilefirstplatformfoundationpush:8.0.+'
```

2. In **Android → app → manifests**, open the `AndroidManifest.xml` file.
   - Add the following permissions to the top the `manifest` tag:

     ```
     <!-- Permissions -->
     <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" />
     ```

   - Add the following (`MFPPush Intent Service`, `MFPPush Instance ID Listener Service`) to the `application` tag:

```
<!-- 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.MFPPushIntentSer
vice"
   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.MFPPushInstanceI
DListenerService"
   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.

Notifications API

MFPPush Instance

All API calls must be called on an instance of `MFPPush`. This can be by created a class level field such as `private MFPPush push = MFPPush.getInstance();`, and then calling `push.<api-call>` throughout the class.

Alternatively you can call `MFPPush.getInstance().<api_call>` for each instance in which you need to access the push API methods.

Challenge Handlers

If the `push.mobileclient` scope is mapped to a **security check**, you need to make sure matching **challenge handlers** exist and are registered before using any of the Push APIs.

> Learn more about challenge handlers in the credential validation (../../authentication-and-security/credentials-validation/android) tutorial.

Client-side

Java Methods	Description
`initialize(Context context);`	Initializes MFPPush for supplied context.
`isPushSupported();`	Does the device support push notifications.
`registerDevice(JSONObject, MFPPushResponseListener);`	Registers the device with the Push Notifications Service.
`getTags(MFPPushResponseListener)`	Retrieves the tag(s) available in a push notification service instance.
`subscribe(String[] tagNames, MFPPushResponseListener)`	Subscribes the device to the specified tag(s).
`getSubscriptions(MFPPushResponseListener)`	Retrieves all tags the device is currently subscribed to.
`unsubscribe(String[] tagNames, MFPPushResponseListener)`	Unsubscribes from a particular tag(s).
`unregisterDevice(MFPPushResponseListener)`	Unregisters the device from the Push Notifications Service

Initialization

Required for the client application to connect to MFPPush service with the right application context.

- The API method should be called first before using any other MFPPush APIs.
- Registers the callback function to handle received push notifications.

```
MFPPush.getInstance().initialize(this);
```

Is push supported

Checks if the device supports push notifications.

```
Boolean isSupported = MFPPush.getInstance().isPushSupported();

if (isSupported ) {
   // Push is supported
} else {
   // Push is not supported
}
```

Register device

Register the device to the push notifications service.

```
MFPPush.getInstance().registerDevice(null, new MFPPushResponseListener<String>() {
   @Override
   public void onSuccess(String s) {
      // Successfully registered
   }

   @Override
   public void onFailure(MFPPushException e) {
      // Registration failed with error
   }
});
```

Get tags

Retrieve all the available tags from the push notification service.

```
MFPPush.getInstance().getTags(new MFPPushResponseListener<List<String>>() {
    @Override
    public void onSuccess(List<String> strings) {
        // Successfully retrieved tags as list of strings
    }

    @Override
    public void onFailure(MFPPushException e) {
        // Failed to receive tags with error
    }
});
```

Subscribe

Subscribe to desired tags.

```
String[] tags = {"Tag 1", "Tag 2"};

MFPPush.getInstance().subscribe(tags, new MFPPushResponseListener<String[]>() {
    @Override
    public void onSuccess(String[] strings) {
        // Subscribed successfully
    }

    @Override
    public void onFailure(MFPPushException e) {
        // Failed to subscribe
    }
});
```

Get subscriptions

Retrieve tags the device is currently subscribed to.

```
MFPPush.getInstance().getSubscriptions(new MFPPushResponseListener<List<String>>() {
    @Override
    public void onSuccess(List<String> strings) {
        // Successfully received subscriptions as list of strings
    }

    @Override
    public void onFailure(MFPPushException e) {
        // Failed to retrieve subscriptions with error
    }
});
```

Unsubscribe

Unsubscribe from tags.

```
String[] tags = {"Tag 1", "Tag 2"};

MFPPush.getInstance().unsubscribe(tags, new MFPPushResponseListener<String[]>() {
    @Override
    public void onSuccess(String[] strings) {
        // Unsubscribed successfully
    }

    @Override
    public void onFailure(MFPPushException e) {
        // Failed to unsubscribe
    }
});
```

Unregister

Unregister the device from push notification service instance.

```
MFPPush.getInstance().unregisterDevice(new MFPPushResponseListener<String>() {
    @Override
    public void onSuccess(String s) {
        disableButtons();
        // Unregistered successfully
    }

    @Override
    public void onFailure(MFPPushException e) {
        // Failed to unregister
    }
});
```

Handling a push notification

In order to handle a push notification you will need to set up a `MFPPushNotificationListener`. This can be achieved by implementing one of the following methods.

Option One

In the activity in which you wish the handle push notifications.

1. Add `implements MFPPushNofiticationListener` to the class declaration.
2. Set the class to be the listener by calling `MFPPush.getInstance().listen(this)` in the `onCreate` method.
3. Then you will need to add the following *required* method:

   ```
   @Override
   public void onReceive(MFPSimplePushNotification mfpSimplePushNotification) {
       // Handle push notification here
   }
   ```

4. In this method you will receive the `MFPSimplePushNotification` and can handle the notification for the desired behavior.

Option Two

Create a listener by calling `listen(new MFPPushNofiticationListener())` on an instance of `MFPPush` as outlined below:

```
MFPPush.getInstance().listen(new MFPPushNotificationListener() {
    @Override
    public void onReceive(MFPSimplePushNotification mfpSimplePushNotification) {
        // Handle push notification here
    }
});
```

Sample application

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/PushNotificationsAndroid/tree/release80) the Android Studio project.

Sample usage

Follow the sample's README.md file for instructions.