Push notifications in native Android applications

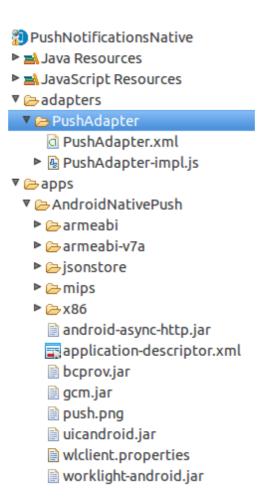
Overview

This tutorial explains the concept, API, and usage of push notification in the context of native Android applications.

The following topics are covered:

- Setting up for push notification
- · Server-side notification APIs
- Client-side notification APIs
- Tag-based notification
- Broadcast-based notification
- Sample application

Setting up your native Android application for push notification



In MobileFirst Studio, create a MobileFirst project and add a MobileFirst Android Native API and HTTP adapter.

The native API includes the following files:

- The *gcm.jar* file contains classes that are necessary for the Android application to register with Google Cloud Messaging (GCM).
- The *push.png* file is an icon file that is displayed when a push notification arrives.

Edit the application-descriptor.xml file.

 Replace the key and senderId values with your API key and project number respectively in the pushSender tag.

In case you do not have an API key, you can get one from the Google GCM Console (https://code.google.com/apis/console)

- Your project's number is the senderId
- Your Android key is the GCM key (can be generated in API & Auth > Credentials)

<nativeAndroidApp xmIns="http://www.worklight.com/native-android-descriptor" id="AndroidNativePus h" platformVersion="6.3.0.00.20141105-0943" securityTest="MySecurityTest" version="1.0"> <pushSender key="SOME-GCM-KEY" senderId="SOME-GCM-ID"/> </nativeAndroidApp>

Deploy the native API and the adapter.

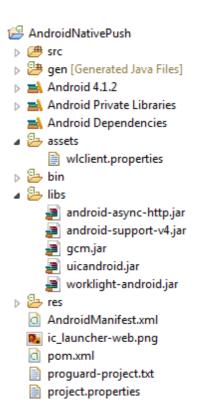
Right-click the generated NativeAPI and select Run As > Deploy Native API.

Copy libraries from the MobileFirst native API to the Android project.

• Copy the following files from the MobileFirst project to the Android project.

MobileFirst project	Android Activity project
wlclient.properties	assets/wlclient.properties
gcm.jar	libs/gcm.jar
worklight-android.jar	libs/worklight-android.jar
android-async-http.jar	libs/android-async-http.jar
push.png	drawable*/push.png

• After you copy the libraries, the native Android project appears as shown:



Edit the wlclient.properties file.

Edit the wlclient.properties file in your native Android project and enter appropriate values for the following fields:

- wlServerHost Hostname or IP address of MobileFirst Server.
- wlServerPort Port on which MobileFirst Server is listening.
- wlServerContext Context root of your MobileFirst Server instance.
- GcmSenderld The project number that you obtained through the Google API console.

```
wlServerProtocol = http
wlServerHost =
wlServerPort = 10080
wlServerContext = /PushNotificationsNative/
wlAppId = AndroidNativePush
wlAppVersion = 1.0
wlEnvironment = Androidnative
wlUid = wY/mbnwKTDDYQUvuQCdSgg==
wlPlatformVersion = 6.3.0.00.20141012-0730
#languagePreferences = Add locales in order of preference (e.g. en, fr, fr-CA)
#For Push Notifications,uncomment below line and assign value to it
GcmSenderId =
```

Modifications to the native Android project

Add the following permissions to the AndroidManifest.xml file of your Android project:

Add the launchMode attribute to the main AndroidNativePush activity. Set its value to singleTask.

```
<activity
android:name="com.worklight.androidnativepush.AndroidNativePush"

android:label="@string/app_name"
android:theme="@android:style/Theme.Black.NoTitleBar"
android:launchMode="singleTask">
```

Add an intent-filter to the main AndroidNativePush activity for notifications.

```
<intent-filter>
    <action android:name="com.worklight.androidnativepush.AndroidNativePush.NOTIFICATION" /
>
    <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
```

Add the GCMIntentService and add an intent-filter for RECEIVE and REGISTRATION of notifications.

Notification API: Server side

Creating an event source.

This can be achieved by creating a notification event source in the adapter JavaScript[™] code at a global level (outside any JavaScript function).

```
WL.Server.createEventSource({
    name: 'PushEventSource',
    onDeviceSubscribe: 'deviceSubscribeFunc',
    onDeviceUnsubscribe: 'deviceUnsubscribeFunc',
    securityTest:'PushApplication-strong-mobile-securityTest
'
});
```

- name A name by which the event source is referenced.
- onDeviceSubscribe An adapter function that is called when the request for user subscription is received.
- onDeviceUnsubscribe An adapter function that is called when the request for user unsubscription is received.
- securityTest A security test from the authenticationConfig.xml file that is used to protect the event source.

Sending a notification

Notifications can be either polled from, or pushed by, the back-end system. In this example, a submitNotifications() adapter function is invoked by a back-end system as an external API to send notifications.

```
function submitNotification(userId, notificationText) {
   var userSubscription = WL.Server.getUserNotificationSubscription('PushAdapter.PushEventSource', userId);
   if (userSubscription === null) {
      return { result: "No subscription found for user :: " + userId };
   }
   var badgeDigit = 1;
   var notification = WL.Server.createDefaultNotification(notificationText, badgeDigit, {custom:"data"});
   WL.Server.notifyAllDevices(userSubscription, notification);
   return {
      result: "Notification sent to user :: " + userId
   };
}
```

Notification API - Client side

The first step is to create an instance of the WLClient class:

```
final WLClient client = WLClient.createInstance(this);
```

You derive all push notification operations from the WLPush class.

getPush — Use this method to retrieve an instance of the WLPush class from the WLClient instance.

```
WLPush push = client.getPush();
```

WL0nReadyToSubscribeListener – When connecting to MobileFirst Server, the application attempts to register itself with the GCM server to receive push notifications.

```
client.getPush().setOnReadyToSubscribeListener(listener);
client.connect(listener);
```

The onReadyToSubscribe method of WLOnReadyToSubscribeListener is called when the registration is complete.

```
@Override
public void onReadyToSubscribe() {.....}
```

WLPush.registerEventSourceCallback

 $To\ register\ an\ a lias\ on\ a\ particular\ event\ source,\ use\ the\ \ WLPush.register EventSource Callback\ method.$

The API takes the following arguments:

```
alias - An alias name.
```

Adaptername - Adapter in which the event source is defined.

EventSourceName - The event source on which the alias is called.

Example:

```
WLClient.getInstance().getPush().registerEventSourceCallback("myAndroid","PushAdapter","PushEventSource",this);
```

Typically, this method is called in the onReadyToSubscribe callback function.

```
@Override
public void onReadyToSubscribe() {
   WLClient.getInstance().getPush().registerEventSourceCallback("myAndroid","PushAdapter","PushEventSource",this);
}
```

In the Android activity class, override the methods that define the Android activity life cycle as follows:

onPause() must call the setForeground(false) method of the WLPush instance to receive the notification in the notification bar when the application is paused.

```
@Override
protected void onPause() {
    super.onPause();
    if (push != null)
    push.setForeground(false);
}
```

onResume() must call the setForeground(true) method of the WLPush instance to receive the notification in the callback of the application.

```
@Override
protected void onResume() {
    super.onResume();
    if (push != null)
    push.setForeground(true);
}
```

onDestroy() must call the unregisterReceivers method of the WLPush instance to avoid leak exceptions from the receiver when the application exits.

```
@Override
protected void onDestroy() {
    super.onDestroy();
    if (push != null)
    push.unregisterReceivers();
}
```

Subscribing to push notifications

To set up subscription to push notifications, use the WLPush.subscribe(alias, pushOptions, responseListener) API.

The API takes the following arguments:

```
alias — The alias to which the device must subscribe.

pushOptions — An object of type WLPushOptions.

responseListener — An object of type WLResponseListener, which is called when subscription completes.
```

Example:

```
WLClient client = WLClient.getInstance(); client.getPush().subscribe("myAndroid",new WLPushOptions(), new MyListener(MyListener.MODE_SUBSC RIBE));
```

MyListener Implements WLResponseListener and provides the following callback functions:

```
onSuccess – Called when subscription succeeds. onFailure – Called when subscription fails.
```

Unsubscribing from push notifications

To set up unsubscription from push notifications, use the WLPush.unsubscribe(alias, responseListener) API.

The API takes the following arguments:

alias — The alias to which the device has subscribed. responseListener — An object of type WLResponseListener, which is called when unsubscription completes.

Example:

```
WLClient client = WLClient.getInstance();
client.getPush().unsubscribe("myAndroid",new MyListener(MyListener.MODE_UNSUBSCRIBE));
```

MyListener Implements WLResponseListener and provides the following callback functions:

```
onSuccess – Called when unsubscription succeeds. onFailure – Called when unsubscription fails.
```

Additional client-side API methods:

isPushSupported() - Indicates whether push notifications are supported by the device.

```
WLClient client = WLClient.getInstance();
boolean supported = client.getPush().isPushSupported();
```

isSubscribed() - Indicates whether the device is subscribed to push notifications.

```
WLClient client = WLClient.getInstance();
boolean blsSubscribed = client.getPush().isSubscribed("myAndroid");
```

Receiving a push notification

When a push notification is received, the onReceive method is called on an WLEventSourceListener instance.

public class MyListener implements WLOnReadyToSubscribeListener, WLResponseListener, WLEventS ourceListener{

The WLEventSourceListener instance is registered during the registerEventSourceCallback callback.

```
WLClient.getInstance().getPush().registerEventSourceCallback("myAndroid", "PushAdapter", "PushEventSource", this );
```

The onReceive method displays the received notification on the screen.

```
@Override
public void onReceive(String arg0, String arg1) {
    AndroidNativePush.updateTextView("Notification received " +
    arg0);
}
```

If the application is not running, the notification icon appears on the notification bar at the top of the screen

Notification API - Tag-based notification

This notification type enables sending and receiving messages by tags.

Tags represent topics of interest to the user and provide the ability to receive notifications according to the chosen interest.

Tags are defined in application-descriptor.xml:

Client-side API methods:

- 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.

Notification API - Broadcast notification

Broadcast notifications are enabled by default for any push-enabled MobileFirst application. A subscription to a reserved tag, Push.ALL, is created for every device.

You can disable broadcast notifications by unsubscribing from the reserved tag Push.ALL

For more information about broadcast notification, see the topic about broadcast notification in the user documentation.

Common APIs for tag-based and broadcast notification

Client-side API:

• 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.

Server-side API:

This method takes two mandatory parameters:

WL.Server.sendMessage(applicationId,notificationOptions)

applicationId - (mandatory) The name of the MobileFirst application. **notificationOptions** - (mandatory) A JSON block containing message properties.

Submits a notification that is based on the specified target parameters.

For a full list of message properties, see the user documentation.

Sample application

Click to download

(http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v630/PushNotificationsNativeProject.zip) the Studio project.

Click to download

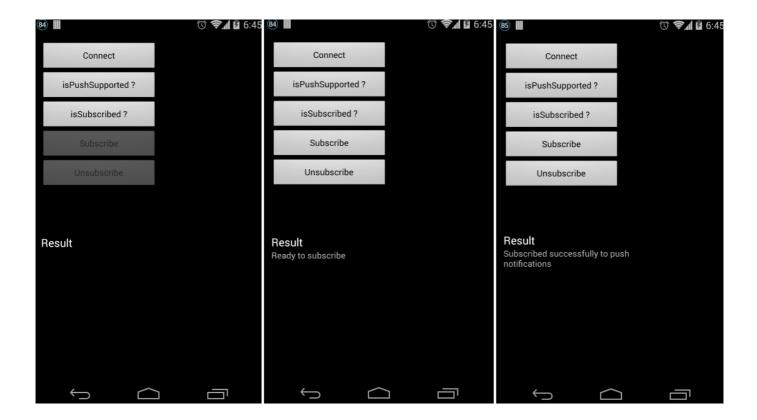
(http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v630/AndroidNativePushProject.zip) the Native project.

The sample contains two projects:

- -The **PushNotificationsNativeProject.zip** file contains a MobileFirst native API that you can deploy to your MobileFirst Server instance.
- -The **AndroidNativePushProject.zip** file contains a native Android application that uses a MobileFirst native API library to subscribe to push notifications and receive notifications from GCM.

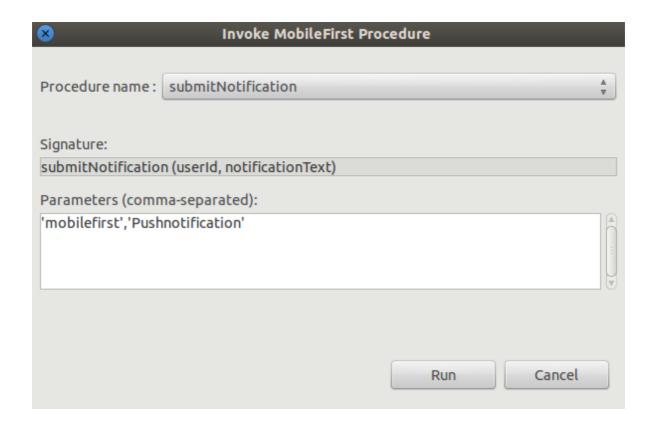
Make sure to update the wlclient.properties file in AndroidNativePushProject with the relevant server settings.

To run the Android application, select Run As > Android Application.



In MobileFirst Studio, right-click **Push Adapter** and select **Run As > Invoke MobileFirst Procedure**.

Call submitNotification to send a push notification.



Push notification received - application in background



