Event source-based notifications in native Windows 8 applications

fork and edit tutorial (https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/7.0/notifications/push-notifications-native-android-applications/event-source-based-notifications.html) | report issue (https://github.ibm.com/MFPSamples/DevCenter/issues/new)

Overview

Event source notifications are notification messages that are targeted to devices with a user subscription. To learn more about the architecture and terminology of push notifications in MobileFirst Platform, refer to the "Event source-based notifications in hybrid applications (../../push-notifications-hybrid-applications/event-source-based-notifications/)" tutorial. Go to:

- Notification API Server-side
- Notification API Client-side
- Sample application

Notification API: Server-side

Creating an event source

Create 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 pulled 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();
```

WLOnReadyToSubscribeListener – 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 alias on a particular event source, use the WLPush.registerEventSourceCallback 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:

```
WLC lient.getInstance ().getPush ().register EventSource Callback ("myAndroid", "PushAdapter", "PushEventSource", \ce", \cdots and \cdots and
```

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.

 $\textbf{public class} \ \textbf{MyListener implements} \ \textbf{WLOnReadyToSubscribeListener}, \ \textbf{WLResponseListener}, \ \textbf{WLEventSourceListener} \\ \textbf{ULResponseListener} \\ \textbf{WLEventSourceListener} \\ \textbf{WLResponseListener} \\ \textbf{WLEventSourceListener} \\ \textbf{WLEventSourceListener} \\ \textbf{WLResponseListener} \\ \textbf{WLEventSourceListener} \\ \textbf{WLResponseListener} \\ \textbf{WLEventSourceListener} \\ \textbf{WLResponseListener} \\ \textbf{WLEventSourceListener} \\ \textbf{WLResponseListener} \\ \textbf{WLResponseListener}$

The WLEventSourceListener instance is registered during the registerEventSourceCallback callback.

 $WLClient.getInstance().getPush().registerEventSourceCallback("myAndroid", "PushAdapter", "PushEventSource", \\ \textbf{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.

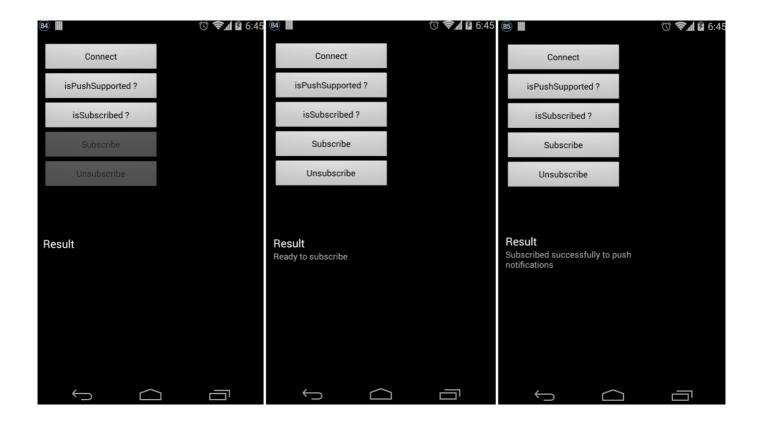
Sample application

Click to download

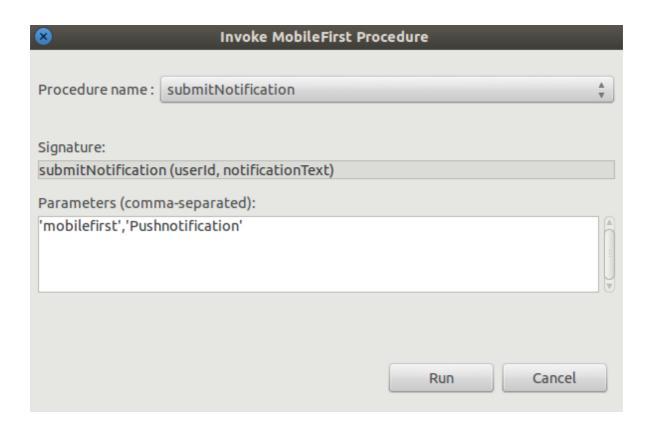
(http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v700/PushNotificationsNativeProject.zip) the Studio project. Click to download

(http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v700/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 iOS 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.



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



