

Event source-based notifications in native Windows 8 applications

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).

```
1
2 WL.Server.createEventSource({
3   name: 'PushEventSource',
4   onDeviceSubscribe: 'deviceSubscribeFunc',
5   onDeviceUnsubscribe: 'deviceUnsubscribeFunc',
6   securityTest: 'PushApplication-strong-mobile-securityTest'
7 });
```

- 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.

```

1
2 function submitNotification(userId, notificationText) {
3     var userSubscription = WL.Server.getUserNotificationSubscription('PushAdapter.PushEventSource', userId);
4
5     if (userSubscription === null) {
6         return { result: "No subscription found for user :: " + userId };
7     }
8
9     var badgeDigit = 1;
10    var notification = WL.Server.createDefaultNotification(notificationText, badgeDigit, {custom:"data"});
11
12    WL.Server.notifyAllDevices(userSubscription, notification);
13
14    return {
15        result: "Notification sent to user :: " + userId
16    };
17 }

```

Notification API - Client-side

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

```

1 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.

```

1 WLPush push = client.getPush();

```

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

```

1
2 client.getPush().setOnReadyToSubscribeListener(listener);
3 client.connect(listener);

```

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

```

1
2 @Override
3 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:

```
1  
2 WLCClient.getInstance().getPush().registerEventSourceCallback("myAndroid","PushAdapter","PushEventSource"
```

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

```
1 @Override  
2 public void onReadyToSubscribe() {  
3     WLCClient.getInstance().getPush().registerEventSourceCallback("myAndroid","PushAdapter","PushEventSource"  
4 }
```

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.

```
1  
2 @Override  
3 protected void onPause() {  
4     super.onPause();  
5     if (push != null)  
6         push.setForeground(false);  
7 }
```

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

```
1  
2 @Override  
3 protected void onResume() {  
4     super.onResume();  
5     if (push != null)  
6         push.setForeground(true);  
7 }
```

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

```

1
2  @Override
3  protected void onDestroy() {
4      super.onDestroy();
5      if (push != null)
6          push.unregisterReceivers();
7  }

```

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:

```

1
2  WLClient client = WLClient.getInstance();
3  client.getPush().subscribe("myAndroid", new WLPushOptions(), new MyListener(MyListener.MODE_SUBSCRI

```

`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:

```

1
2  WLClient client = WLClient.getInstance();
3  client.getPush().unsubscribe("myAndroid", new MyListener(MyListener.MODE_UNSUBSCRIBE));
4

```

`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.

```

1
2  WLClient client = WLClient.getInstance();
3  boolean supported = client.getPush().isPushSupported();

```

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

```
1
2  WLClient client = WLClient.getInstance();
3  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.

```
1
2  public class MyListener implements WLOnReadyToSubscribeListener, WLResponseListener, WLEventSou
```

The `WLEventSourceListener` instance is registered during the `registerEventSourceCallback` callback.

```
1
2  WLClient.getInstance().getPush().registerEventSourceCallback("myAndroid", "PushAdapter", "PushEventSource
```

The `onReceive` method displays the received notification on the screen.

```
1
2  @Override
3  public void onReceive(String arg0, String arg1) {
4      AndroidNativePush.updateTextView("Notification received " + arg0);
5  }
```

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



Push notification received - application in foreground

