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-windows-phone-8-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 IBM MobileFirst™ Platform Foundation, see the “Event source-based notifications in hybrid applications (../../push-notifications-hybrid-applications/event-source-based-notifications/)” tutorial.

For more information about setting up push notifications in native Windows 8 applications, see the “ Push notifications in native Windows 8 applications (../)” 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, which 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', userI
d);
    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:

```
WLClient client = WLClient.getInstance();
```

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 Google Cloud Messaging (GCM) server to receive push notifications.

```
OnReadyToSubscribeListener myOnReadyListener = new OnReadyToSubscribeListener();
push.onReadyToSubscribeListener = myOnReadyListener;
```

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

```
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:

```
WLClient.getInstance().getPush().registerEventSourceCallback("myPush", "PushAdapter", "PushEventSourc
e", this);
```

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

```
public void onReadyToSubscribe()
{
  WLClient.getInstance().getPush().registerEventSourceCallback("myPush", "PushAdapter","PushEventSourc
e", this);
}
```

Subscribing to push notification

To set up subscription to push notification, 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:

```
WLPush push = WLClient.getInstance().getPush();
MySubscribeListener mySubListener = new MySubscribeListener();
push.subscribe("myPush", null, mySubListener);
```

`MySubscribeListener` 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 notification, 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:

```
WLPush push = WLClient.getInstance().getPush();
MyUnsubscribeListener myUnsubListener = new MyUnsubscribeListener();
push.unsubscribe("myPush", myUnsubListener);
```

`MyUnsubscribeListener` implements `WLResponseListener` and provides the following callback functions:

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

Additional client-side API methods

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

```
WLClient.getInstance().getPush().isSubscribed("myPush");
```

Receiving a push notification

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

```
class OnReadyToSubscribeListener : WLOnReadyToSubscribeListener, WLEventSourceListener{...}
```

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

```
WLClient.getInstance().getPush().registerEventSourceCallback("myPush", "PushAdapter", "PushEventSourc
e", this);
```

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

```
public void onReceive(String props, String payload)
{
   Debug.WriteLine("Props: " + props);
   Debug.WriteLine("Payload: " + payload);
}
```

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/Windows8NativePushProject.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 `Windows8NativePushProject.zip` file contains a **native Windows 8 application** that uses a MobileFirst native API library to subscribe to push notifications and receive notifications from Windows Notification Services (WNS).
  Make sure to update the `wlclient.properties` file in Windows8NativePushProject with the relevant server settings.