# 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. Jump to:

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

While the user subscription exists, MobileFirst Server can produce push notifications for the subscribed user. These notifications can be delivered by the adapter code to all or some of the devices from which the user subscribed. Implementation of the push notification API consists of the following main steps:

### On the server side:

- *Creating an event source*
- *Sending notification*

### On the client side:

- *Sending the token and initializing the* `WLPush` *class*
- *Registering the event source*
- *Subscribing to/unsubscribing from the event source*

## 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', 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

A device subscription belongs to a user subscription and exists in the scope of a specific user and event source. A user subscription can have several device subscriptions. The device subscription is created when the application on a device calls the `[[WLPush sharedInstance]subscribe]` method. The device subscription is deleted either by an application that calls `[[WLPush sharedInstance] unsubscribe]` or when the push mediator informs MobileFirst Platform Server that the device is permanently not accessible.

1. Access the `WLPush` functionality by using `[WLPush sharedInstance]` anywhere in your application.
2. Create an instance of `onReadyToSubscribeListener` and set values for alias, adaptername and eventsource.

   ```
   ReadyToSubscribeListener *readyToSubscribeListener = [[ReadyToSubscribeListener alloc] initWith
   Controller:self];
     readyToSubscribeListener.alias = self.alias;
     readyToSubscribeListener.adapterName = self.adapterName;
     readyToSubscribeListener.eventSourceName = self.eventSourceName;
   ```

3. Set the `onReadyToSubscribeListener` on `WLPush`.

   ```
   [[WLPush sharedInstance] setOnReadyToSubscribeListener:readyToSubscribeListener];
   ```

4. Pass the token to `WLPush`.

   ```
   [[WLPush sharedInstance] setTokenFromClient:deviceToken.description];
   ```

## Sending token to client and initializing `WLPush`

The user must initialize the `WLPush sharedInstance` in the application's `ViewController` load method.

```
AppDelegate *appDelegate = [[UIApplication sharedApplication]delegate];
[[WLPush sharedInstance]init];
```

The user must add this method to the app delegate to get the token.

```
-(void)application:(UIApplication *)application didRegisterForRemoteNotificationsWithDeviceToken:(NSData
*)deviceToken{
}
```

The token that is received by this method must be passed to the WLPush method.

```
[[WLPush sharedInstance] setTokenFromClient:deviceToken.description];
```

## Registering the event source

IBM MobileFirst Platform Foundation provides the customizable `onReadyToSubscribe` function, which is used to register an event source. Set up your `onReadyToSubscribe` function in Listener, which implements `WLOnReadyToSubscribeListener`. This function is called when authentication finishes.

```
#import "ReadyToSubscribeListener.h"
#import "MyEventSourceListener.h"

@implementation ReadyToSubscribeListener

-(void)OnReadyToSubscribe{
  MyEventSourceListener *eventSourceListener=[[MyEventSourceListener alloc]init];
  [[WLPush sharedInstance] registerEventSourceCallback:self.alias
:self.adapterName
  :self.eventSourceName :eventSourceListener];
}

@end
```

## Subscribing to the event source

**Prerequisite:** To subscribe, a user must authenticate. To set up subscription to the event source, use the following API:

```
- (IBAction)subscribe:(id)sender {
  MySubscribeListener *mySubscribeListener = [[MySubscribeListener alloc] initWithController:self]
;
  [[WLPush sharedInstance]subscribe:self.alias :nil :mySubscribeListener];
}
```

`[[WLPush sharedInstance] subscribe]` takes the following parameters:

- An alias, as declared in `[[WLPush sharedInstance] registerEventSourceCallback]`
- `WLPushOptions` - The instance contains the custom subscription parameters that the event source supports (optional).
- `id WLDelegate` - The listener object instance, whose `onSuccess` and `onFailure` callback methods are called (optional).

Delegates receive a response object if one is required.

### Unsubscribing from an event source

To set up unsubscription from an event source, use the following API:

```
- (IBAction)unsubscribe:(id)sender {
  MyUnsubscribeListener *myUnsubscribelistener = [[MyUnsubscribeListener alloc]initWithController:self]
;
  [[WLPush sharedInstance]unsubscribe:self.alias :myUnsubscribelistener];
}
```

`[[WLPush sharedInstance] unsubscribe]` takes the following parameters:

- An alias, as declared in `[[WLPush sharedInstance] registerEventSourceCallback]`
- `id WLDelegate` - The listener object instance, whose `onSuccess` and `onFailure` callback methods are called (optional)

Delegates receive a response object if one is required.

### Additional client-side APIs

`[[WLPush sharedInstance]isPushSupported]` – Returns `true` if push notifications are supported by the platform, or `false` otherwise. `[[WLPush sharedInstance]isSubscribed:alias]` – Returns whether the currently logged-in user is subscribed to a specified event source alias.
When a push notification is received by a device, the `didReceiveRemoteNotification` method is called in the app delegate. Logic to handle the notification should be defined here.

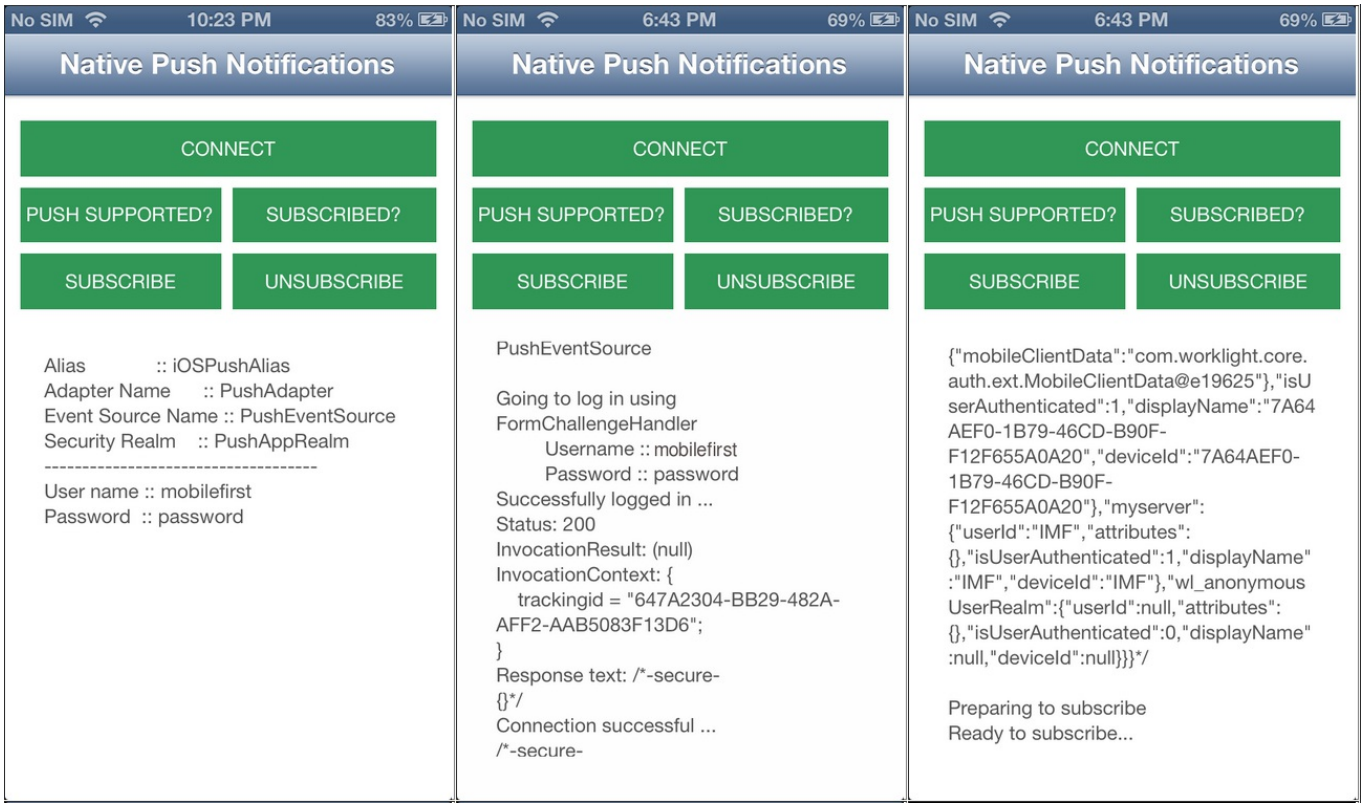
```
-(void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userInfo{
    NSLog(@"Received Notification %@",userInfo.description);
}
```

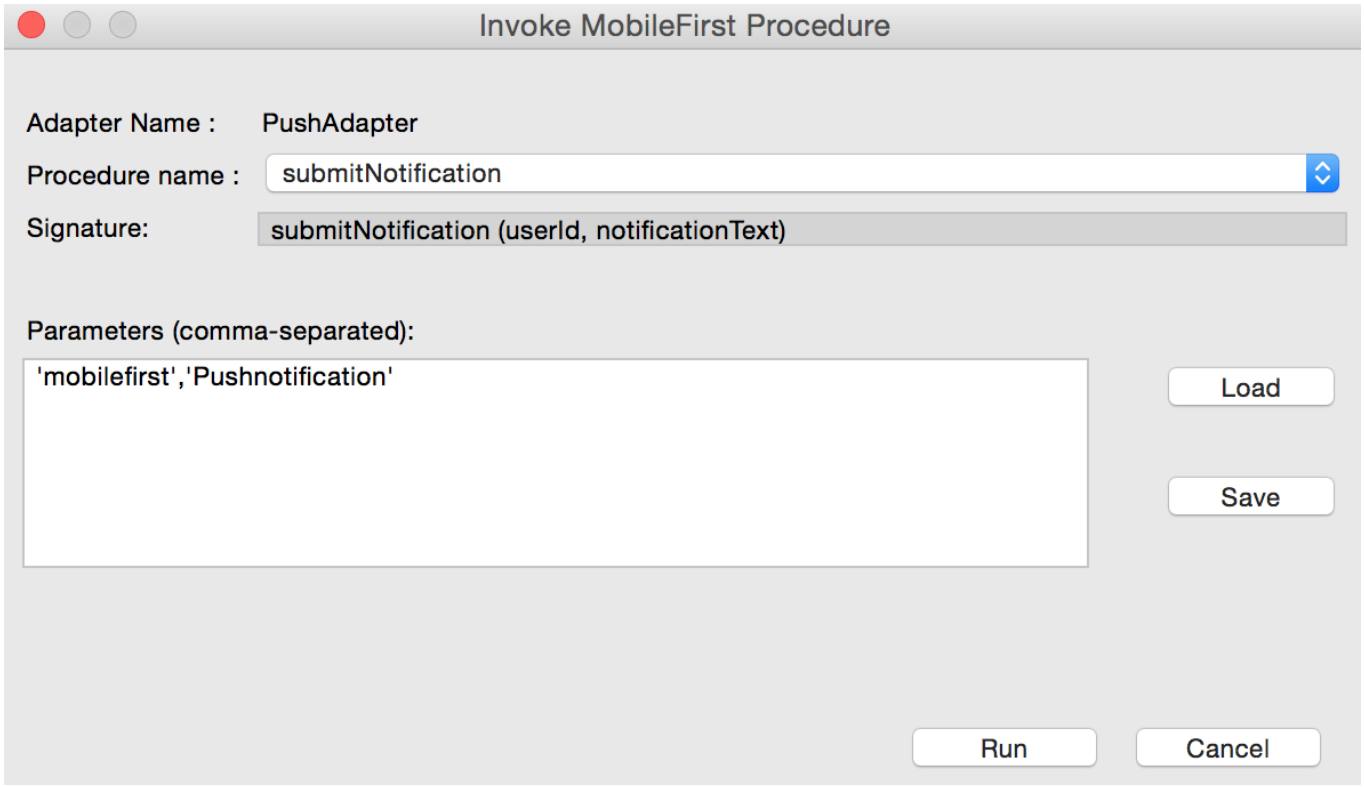
If the application was in background mode (or inactive) when the push notification arrived, this callback is called when the application returns to the foreground.

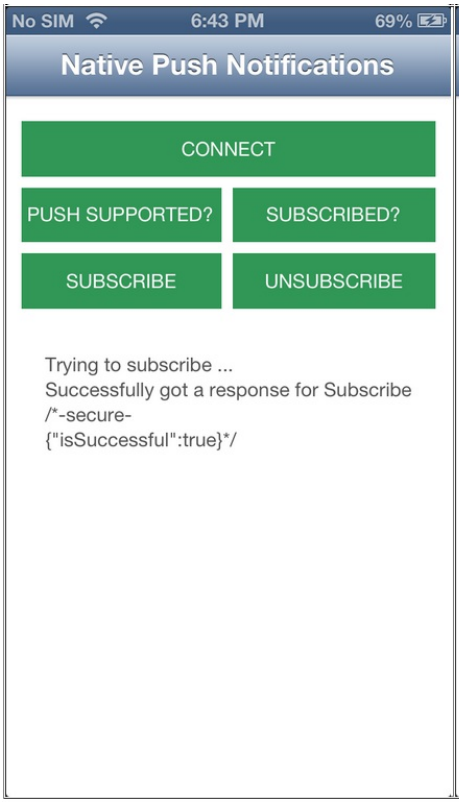
## Sample application

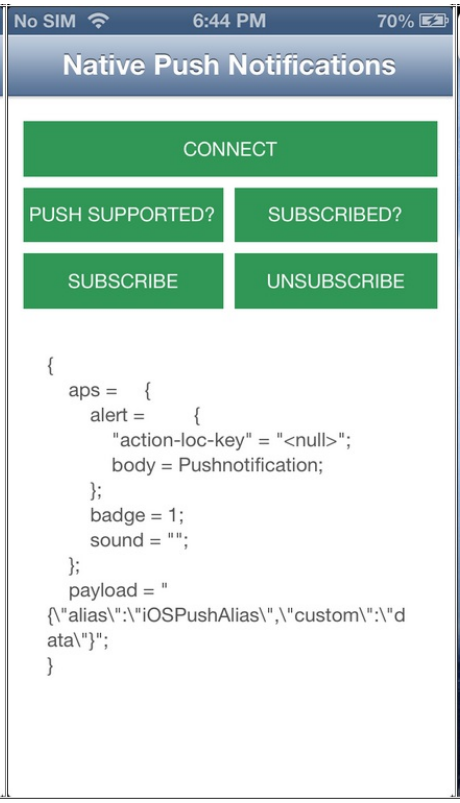
Click to download (http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v700/PushNotificationsNativeProject.zip) the Studio project. Click to download (http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v700/iOSNativePushProject.zip) the Native project. The sample contains two projects:

- The **PushNotificationsNativeProject.zip** file contains a MobileFirst native API that you can deploy to your MobileFirst server.
- **iOSNativePushProject.zip** file contains a native iOS application that uses a MobileFirst native API library to subscribe for push notification and receive notifications from APNS. Make sure to update the **worklight.plist** file in iOSNativePush with the relevant server settings.




Invoke MobileFirst Procedure
Adapter Name :
PushAdapter
Procedure name :
submitNotification
Signature:
submitNotification (userId, notificationText)
Parameters (comma-separated):
'mobilefirst','Pushnotification'
Load
Save
Run
Cancel


No SIM
6:43 PM
69%
Native Push Notifications
CONNECT
PUSH SUPPORTED?
SUBSCRIBED?
SUBSCRIBE
UNSUBSCRIBE
Trying to subscribe ...
Successfully got a response for Subscribe
/*-secure-
{"isSuccessful":true}*/


No SIM
6:44 PM
70%
Native Push Notifications
CONNECT
PUSH SUPPORTED?
SUBSCRIBED?
SUBSCRIBE
UNSUBSCRIBE
{
aps = {
alert = {
"action-loc-key" = "<null>";
body = Pushnotification;
};
badge = 1;
sound = "";
};
payload = "
{\"alias\":\"iOSPushAlias\",\"custom\":\"d
ata\"}";
}


iOSNativePush
Pushnotification
Instagram
Safari
Settings
Chrome