Event Source Notifications in Native iOS Applications

- Download MobileFirst project (https://github.com/MobileFirst-Platform-Developer-Center/EventSourceNotifications)
- Download Native project (https://github.com/MobileFirst-Platform-Developer-Center/EventSourceNotificationsObjC)

Overview

Prerequisite: Make sure to read the Push notifications in native iOS applications (../) tutorial first.

Event source notifications are notification messages that are targeted to devices with a user subscription.
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.

To learn more about the architecture and terminology of event-source push notifications refer to the Push notification overview (../../../push-notifications-overview/#notificationTypes) tutorial.

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

Agenda

- Notification API - server-side
- Notification API - Client-side
- Sample application

Notification API - Server-side

Creating an event source

To create an event source, you declare a notification event source in the adapter JavaScript code at a global level (outside any JavaScript function):

[code lang="js"]WL.Server.createEventSource({
name: 'PushEventSource',
onDeviceSubscribe: 'deviceSubscribeFunc',

onDeviceUnsubscribe: 'deviceUnsubscribeFunc',
securityTest:'PushApplication-strong-mobile-securityTest'
});[/code]

- **name** – a name by which the event source is referenced.
- **onDeviceSubscribe** – an adapter function that is invoked when the user subscription request is received.
- **onDeviceUnsubscribe** – an adapter function that is invoked when the user unsubscription request is received.
- **securityTest** – a security test from the `authenticationConfig.xml` file, which is used to protect the event source.

An additional event source option:
[code lang="js" firstline="6"]poll: {
interval: 3,
onPoll: 'getNotificationsFromBackend'
}[/code]

- **poll** – a method that is used for notification retrieval.
  The following parameters are required:
  - **interval** – the polling interval in seconds.
  - **onPoll** – the polling implementation. An adapter function to be invoked at specified intervals.

Sending a notification

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

[code lang="js"]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
};
}[/code]

The submitNotification function receives the userId to send notification to and the notificationText.

[code lang="js"]function submitNotification(userId, notificationText) {[/code]

A user subscription object contains the information about all of the user's subscriptions. Each user subscription can have several device subscriptions. The object structure is as follows:

[code lang="js" gutter="false"]{
userId: 'bjones',
state: {
customField: 3
},
getDeviceSubscriptions: function()[}
};[/code]

Next line:
[code lang="js" firstline="2"]var userSubscription =
WL.Server.getUserNotificationSubscription('PushAdapter.PushEventSource', userId);[/code]

If the user has no subscriptions for the specified event source, a **null** object is returned.
[code lang="js" firstline="4"]if (userSubscription === null) {
return { result: "No subscription found for user :: " + userId };
}[/code]

The WL.Server.createDefaultNotification API method creates and returns a default notification JSON block for the supplied values.
[code lang="js" firstline="8"]var badgeDigit = 1;
var notification = WL.Server.createDefaultNotification(notificationText, badgeDigit, {custom:"data"});
[/code]

- **notificationText** - The text to be pushed to the device.
- **Badge** (number) - A number that is displayed on the application icon or tile (in environments that support it).
- **custom** - Custom, or Payload, is a JSON object that is transferred to the application and that can contain custom properties.

The WL.Server.notifyAllDevices API method sends notification to all the devices that are subscribed to the user.

[code lang="js" firstline="11"]WL.Server.notifyAllDevices(userSubscription, notification);[/code]

Several APIs exist for sending notifications:

- WL.Server.notifyAllDevices(userSubscription, options) - to send notification to all user's devices.
- WL.Server.notifyDevice(userSubscription, device, options) - to send notification to a specific device that belongs to a specific user subscription.
- WL.Server.notifyDeviceSubscription(deviceSubscription, options) - to send the notification to a specific device.

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.
   [code lang="obj-c"]
   ReadyToSubscribeListener *readyToSubscribeListener = [[ReadyToSubscribeListener alloc] initWithController:self];
   readyToSubscribeListener.alias = self.alias;
   readyToSubscribeListener.adapterName = self.adapterName;
   readyToSubscribeListener.eventSourceName = self.eventSourceName;
   [/code]
3. Set the `onReadyToSubscribeListener` on `WLPush`.
   [code lang="obj-c"]
   [[WLPush sharedInstance] setOnReadyToSubscribeListener:readyToSubscribeListener];
   [/code]
4. Pass the token to WLPush.
   [code lang="obj-c"]
   [[WLPush sharedInstance] setTokenFromClient:deviceToken.description];
   [/code]

Sending token to client and initializing `WLPush`

The user must initialize the `WLPush sharedInstance` in the application's `ViewController` load method.
[code lang="obj-c"]
AppDelegate *appDelegate = [[UIApplication sharedApplication]delegate];
[[WLPush sharedInstance]init];
[/code]
The user must add this method to the app delegate to get the token.
[code lang="obj-c"]
-(void)application:(UIApplication *)application didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken{
}
[/code]

The token that is received by this method must be passed to the WLPush method.
[code lang="obj-c"]
[[WLPush sharedInstance] setTokenFromClient:deviceToken.description];
[/code]

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.
[code lang="obj-c"]
#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
[/code]

Subscribing to the event source

Prerequisite: To subscribe, a user must authenticate.

To set up subscription to the event source, use the following API:
[code lang="obj-c"]
- (IBAction)subscribe:(id)sender {
MySubscribeListener *mySubscribeListener = [[MySubscribeListener alloc] initWithController:self];
[[WLPush sharedInstance]subscribe:self.alias :nil :mySubscribeListener];
}
[/code]

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

- An alias, as declared in `[[WLPush sharedInstance] registerEventSourceCallback]`
- `WLPushOptions` - 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:
[code lang="obj-c"]
- (IBAction)unsubscribe:(id)sender {
MyUnsubscribeListener *myUnsubscribelistener = [[MyUnsubscribeListener

alloc]initWithController:self];
[[WLPush sharedInstance]unsubscribe:self.alias :myUnsubscribelistener];
}
[/code]
[[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.

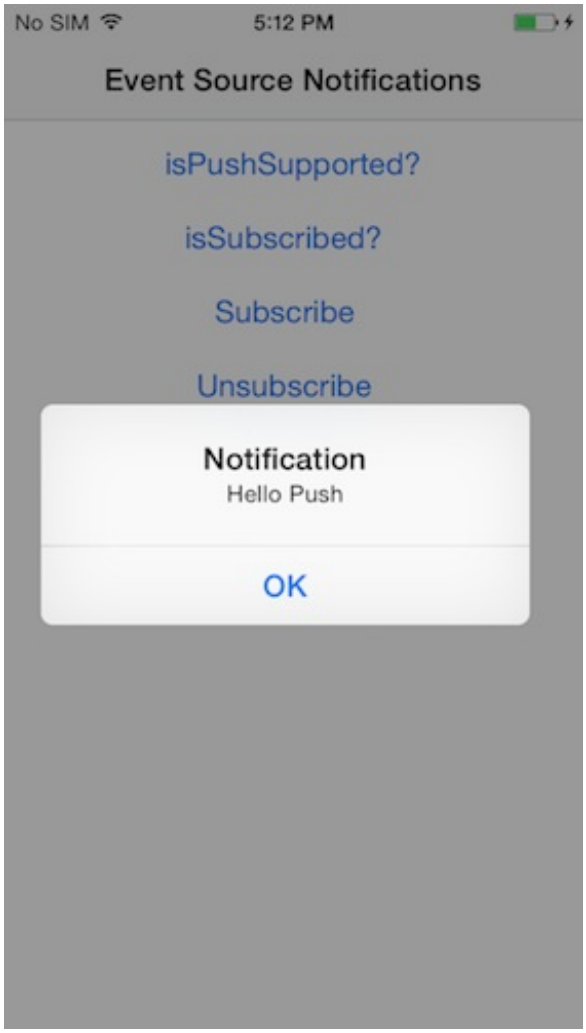
[code lang="obj-c"]
-(void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userInfo{
NSLog(@"Received Notification %@",userInfo.description);
}
[/code]
If the application was in background mode (or inactive) when the push notification arrived, this callback is called when the application returns to the foreground.

Sample application

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/EventSourceNotifications) the MobileFirst project.

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/EventSourceNotificationsObjC) the Native project.

- The `EventSourceNotifications` project contains a MobileFirst native API that you can deploy to your MobileFirst server.
- The `EventSourceNotificationsObjC` project 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 the native project with the relevant server settings.



(https://developer.ibm.com/mobilefirstplatform/wp-content/uploads/sites/32/2015/04/IMG_0014.png)

Sending a notification

To test the application is able to receive a push notification you can perform one of the following:

1. Right-click the adapter in MobileFirst Studio and select **Call MobileFirst Adapter**
2. If using the CLI, for example:
   [code lang="shell"]$ mfp adapter call
   [?] Which endpoint do you want to use? PushAdapter/submitNotification
   [?] Enter the comma-separated parameters: "the-user-name", "hello!"
   [?] How should the procedure be called? GET[/code]