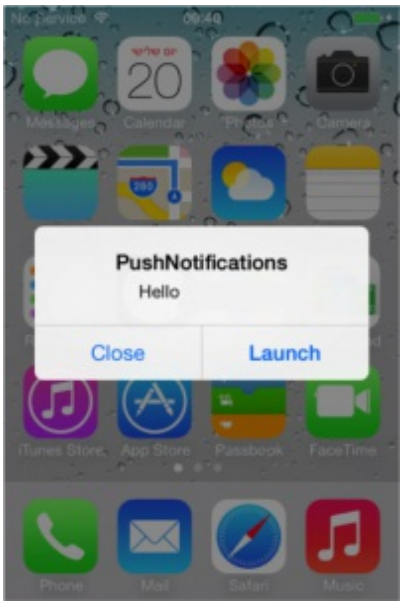
# Event source-based notifications in native Windows 8 applications



## Overview

Event source notifications are notification messages that are targeted to devices with a user subscription. This tutorial explains the concept, API, and usage of push notifications in the context of hybrid applications. The following topics are covered:

- Notification architecture
- Sending notifications
- Subscription management
- Notification API
- Back-end emulator




## Notification architecture

### Terminology

#### Event source

A push notification channel to which mobile applications can register. An event source is defined within a MobileFirst adapter.

#### Device token

A unique identifier, obtained from the push mediator (Apple, Google, or Microsoft), which identifies the request of a specific mobile device to receive notifications from the MobileFirst Server.

#### User ID

A unique identifier for a user. Obtained through authentication or other unique identifier such as a persistent cookie.

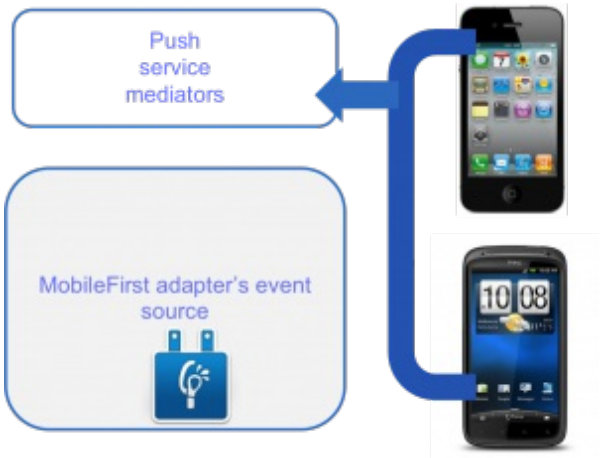
### Application ID

MobileFirst application ID. Identifies a specific MobileFirst application on the mobile market.

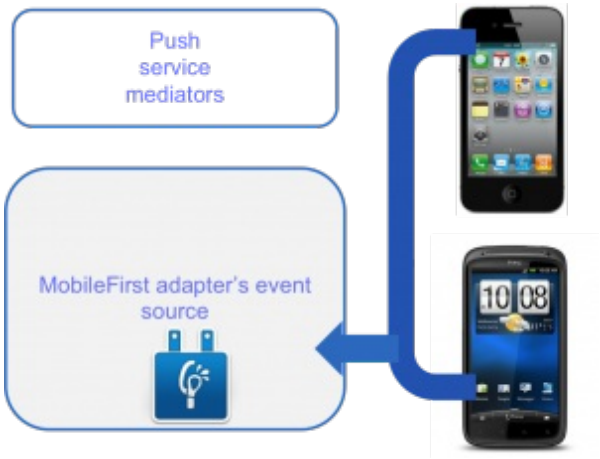
## Subscription

To start receiving push notifications, an application must first subscribe to a push notification event source. An event source is declared in the MobileFirst adapter that is used by the application for push notification services. The end user must approve the push notification subscription. After the subscription is approved, the device registers with an Apple, Google, or Microsoft push server to obtain a token that is used to identify the device (“Allow notifications for application X on device Y”), and sends a subscription request to the MobileFirst Server. *This operation is performed automatically by the MobileFirst framework.*

## Demonstration




When the `subscribe` API method is called, the device registers with a push service mediator and obtains a device token (done automatically by the framework).




When the token is obtained, the application subscribes to an event source. Both actions are done automatically by a single API method call as described later.

# Sending notifications

IBM MobileFirst Platform Foundation provides a unified API for push notification. Use the adapter API for the following actions:

- Managing subscriptions

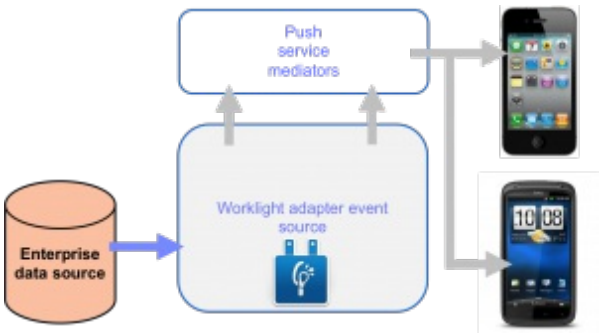
- Pushing and polling notifications from a back end
- Sending push notifications to devices

Use the application API for the following actions:

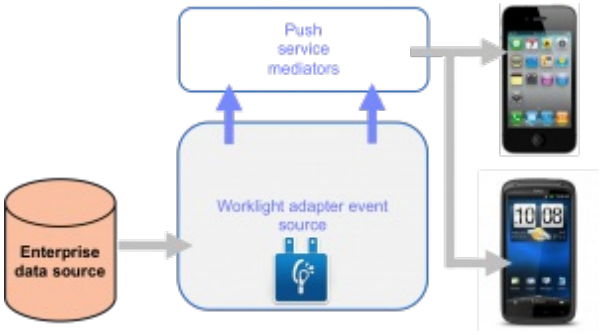
- Subscribing to and unsubscribing from push-notification event sources
- Handling incoming notifications

Before a notification can be sent, it must first be retrieved from the back end. An event source can either **poll** notifications from the back-end system, or wait for the back-end system to explicitly **push** a new notification. When a notification is retrieved by the adapter, it is processed and sent through the corresponding push service mediator (Apple, Google, or Microsoft). Some custom logic can be added in the adapter to preprocess notifications. The push service mediator receives the notification and sends it to a device.

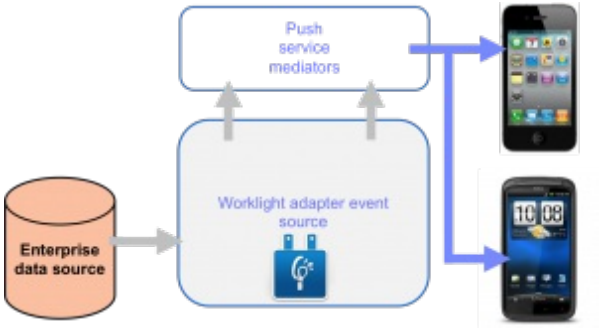
## Demonstration



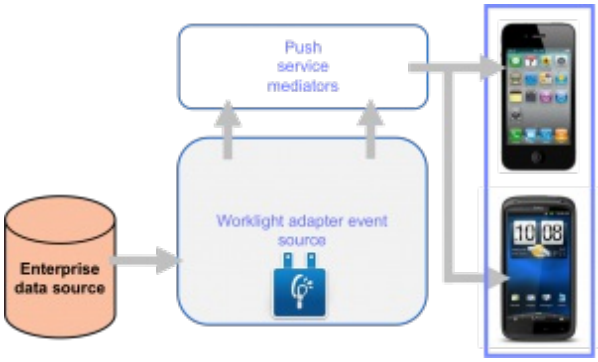

Notifications are retrieved by the MobileFirst adapter event source, either by poll or by push from the back-end system.




The adapter processes the notification and sends it to a push service mediator.




The push service mediator sends a push notification to the device.




The device processes the received notification.

# Subscription management

## User subscription

### Subscription

An entity that contains a user ID, a device ID, and an event source ID. It represents the intent of the user to receive notification from a specific event source.

### Creation

The user subscription for an event source is created when the user first subscribes to the event source from any device.

### Deletion

A user subscription is deleted when the user unsubscribes from the event source from all the user's devices.

### Notification

While the user subscription exists, the 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 that the user subscribed from.

## Device subscription

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 `WL.Client.Push.subscribe()`. The device subscription is deleted either by an application that calls `WL.Client.Push.unsubscribe()`, or when the push mediator informs the MobileFirst Server that the device is permanently not accessible.

# 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):

```
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 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:

```
poll: {
    interval: 3,
    onPoll: 'getNotificationsFromBackend
'
}
```

- **`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.

```
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
    };
}
```

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

```
 function submitNotification(userId, notificationText) {
```

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:

```
{
    userId: 'bjones',
    state: {
        customField: 3
    },
    getDeviceSubscriptions: function()[
}
};
```

Next line:

```
var userSubscription = WL.Server.getUserNotificationSubscription('PushAdapter.PushEventSource', userId);
```

If the user has no subscriptions for the specified event source, a **null** object is returned.

```
if (userSubscription === null) {
    return { result: "No subscription found for user :: " + userId }
;
}
```

The `WL.Server.createDefaultNotification` API method creates and returns a default notification JSON block for the supplied values.

```
var badgeDigit = 1;
var notification = WL.Server.createDefaultNotification(notificationText, badgeDigit, {custom:"data"});
```

- **`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.

```
WL.Server.notifyAllDevices(userSubscription, notification);
```

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

Additional client-side API methods:

- `WL.Client.Push.isPushSupported()` – returns `true` if push notifications are supported by the platform, or `false` otherwise.
- `WL.Client.Push.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 callback function defined in `WL.Client.Push.registerEventSourceCallback` is invoked:

```
function pushNotificationReceived(props, payload) {
    alert("pushNotificationReceived invoked");
    alert("props :: " + JSON.stringify(props));
    alert("payload :: " + JSON.stringify(payload));
}
```

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

# Back-end emulator

The sample project for this tutorial is bundled with a back-end emulator which can be used to simulate push notification submissions by a back-end system. The source for the emulator can be found in the sample project. To run the back-end emulator, open the PushBackendEmulator folder of the sample project in a command prompt, and then run the supplied JAR file by using the following syntax:

```
java –jar PushBackendEmulator.jar <userId> <message> <contextRoot> <port>
```

`userId` is the user name that you used to log in to the sample application. `contextRoot` is the context root of your MobileFirst project.

### Example

```
java –jar PushBackendEmulator.jar JohnDoe “My first push notification” myContextRoot 10080
```

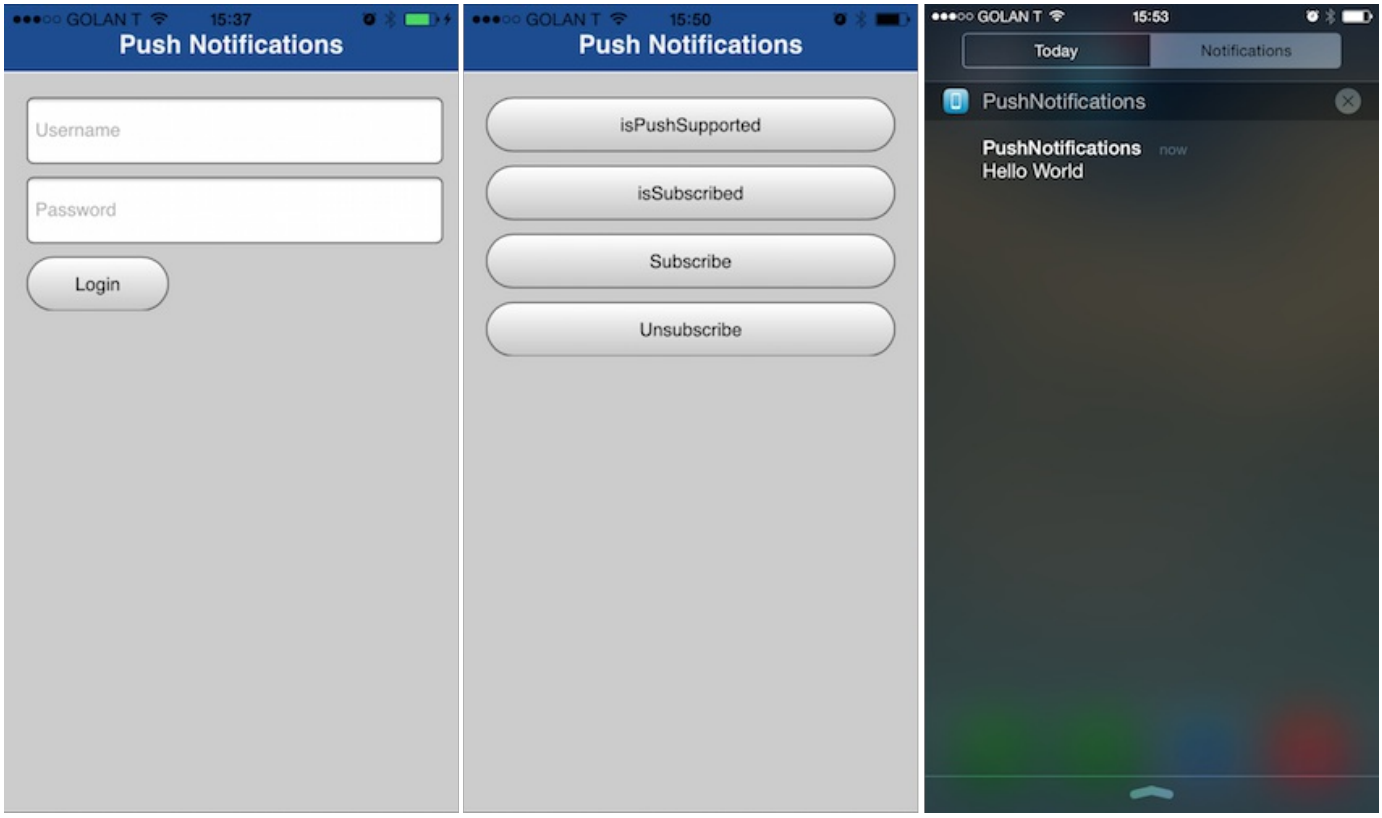
The back-end emulator tries to connect to a MobileFirst Server and call a `submitNotification()` adapter procedure. It outputs the invocation result to a command prompt console.

Success

```
c:\>java -jar C:\PushBackendEmulator.jar JohnDoe "hello push" PushNotificationsProject 10080
PushBackendEmulator
User Id: JohnDoe
Notification text: hello push
Server URL: http://localhost:10080/PushNotificationsProject
sending notification
Server response :: {   "isSuccessful": true,   "result": "Notification sent to user :: JohnDoe"}
```

Failure

```
c:\>java -jar C:\PushBackendEmulator.jar JohnMissing "hello push" PushNotificationsProject 10080
PushBackendEmulator
User Id: JohnMissing
Notification text: hello push
Server URL: http://localhost:10080/PushNotificationsProject
sending notification
Server response :: {   "isSuccessful": true,   "result": "No subscription found for user :: JohnMissing"}
```




## Sample application

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