Push notifications in hybrid applications

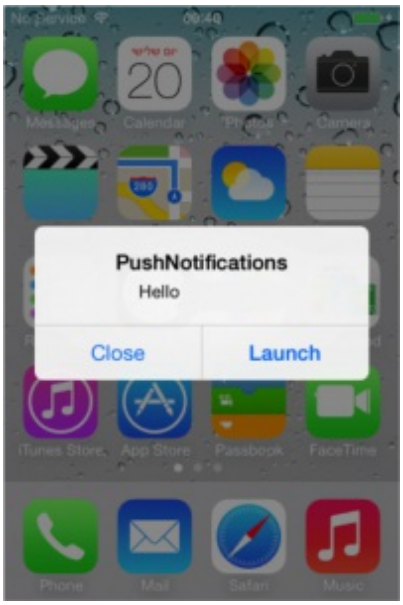
fork and edit tutorial (https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/6.3/notifications/push-notifications-hybrid-applications.html) | report issue (https://github.ibm.com/MFPSamples/DevCenter/issues/new)

Overview

This tutorial explains the concept, API, and usage of push notifications in the context of hybrid applications.

Topics:

- What is push notification?
- Supported devices
- Notification architecture
- Sending notification
- Subscription management
- Notification API - Server side
- Notification API - Client side
- Notification API - Interactive notification
- Notification API - Tag-based notification
- Notification API - Broadcast notification
- Project setup and guidelines
- Back-end emulator
- Sample application



What is push notification?

Push notification is the ability of a mobile device to receive messages that are "pushed" from a server.

Notifications are received regardless of whether the application is currently running.
Notifications can take several forms and are platform-dependent:

- **Alert**: a pop-up text message
- **Badge, Tile**: a graphical representation for a short text or image

- **Banner, Toast**: a disappearing pop-up text message at the top of the device display
- **Sound**

Support devices

IBM MobileFirst Platform Foundation supports push notifications for the following mobile platforms:

- Android 2.3.5, 4.x, 5.x
- iOS 6, 7, and 8
- Windows Phone 8.x

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

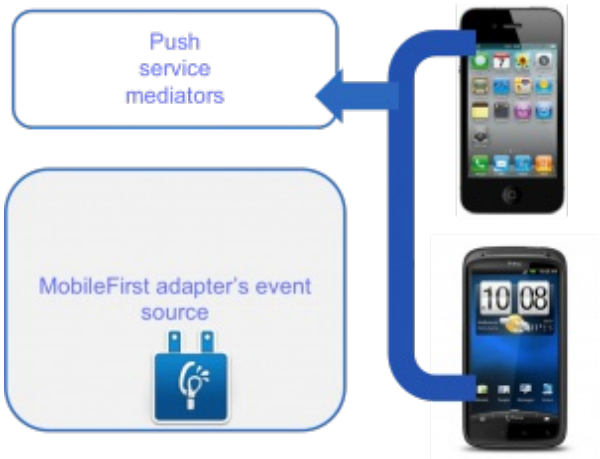
The MobileFirst application ID identifies a specific MobileFirst application on the mobile market.

Subscription

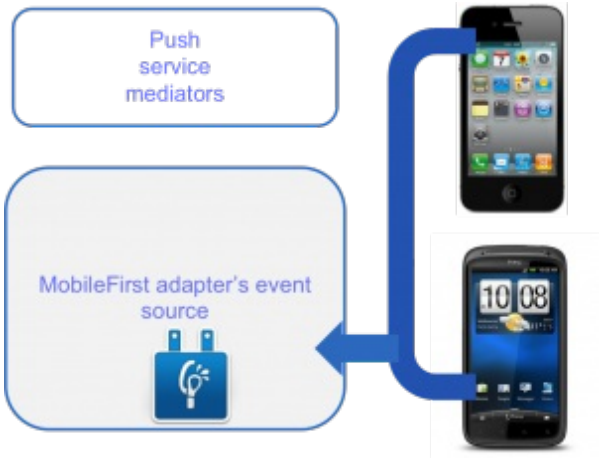
To start receiving push notifications, an application must first subscribe to a push notification event source. An event source is declared in the MobileFirst adapter that is used by the application for push notification services.

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` 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 notification

IBM MobileFirst Platform Foundation provides a unified push notification API.

By using the adapter API, you can:

- Manage subscriptions
- Push and poll notifications from a back-end service
- Send push notifications to devices

By using the application API, you can:

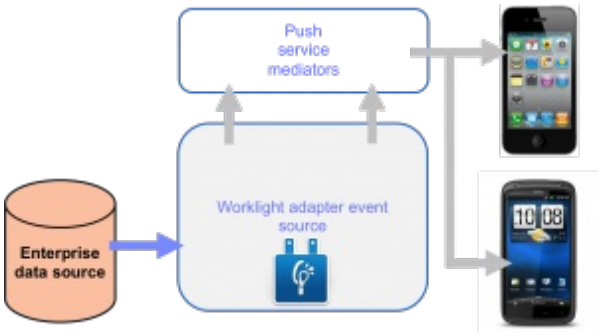
- Subscribe to and unsubscribe from push notification event sources
- Handle incoming notifications

Before a notification is sent, it must first be retrieved from the back end.

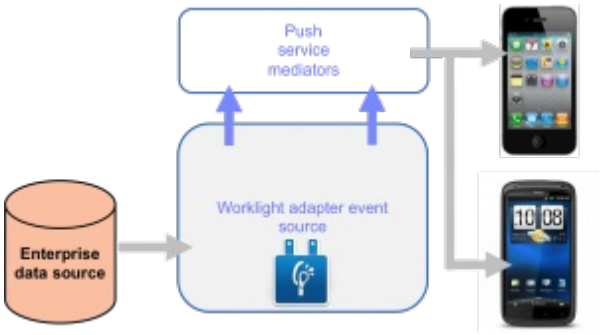
An event source can either **poll** notifications from the back-end system, or wait for the back-end system to explicitly **push** a new notification.
When a notification is retrieved by the adapter, it is processed and sent through the corresponding push service mediator (Apple, Google, or Microsoft).

You can add custom logic in the adapter to preprocess notifications. The push service mediator receives the notification and sends it to a device.

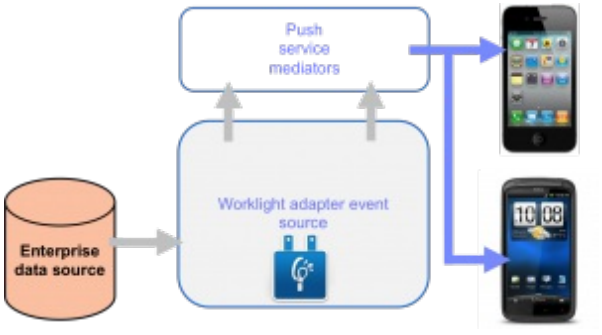
Demonstration



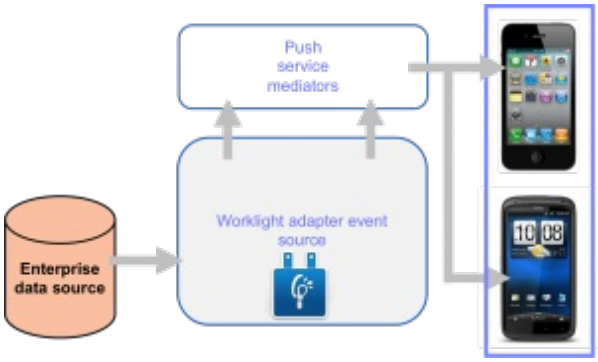
Notifications are retrieved by the MobileFirst adapter event source, they are either by polled from, or by pushed by, the back-end system.



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 the first time the user 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 the `WL.Client.Push.subscribe()` method.
The device subscription is deleted either by an application that calls the `WL.Client.Push.unsubscribe()` method, 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, declare it 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`**: The name by which the event source is referenced
- **`onDeviceSubscribe`**: The adapter function that is invoked when the user subscription request is received
- **`onDeviceUnsubscribe`**: The adapter function that is invoked when the user unsubscription request is received
- **`securityTest`**: A security test from the `authenticationConfig.xml` file that 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 mandatory:
    - **`interval`**: The polling interval in seconds
    - **`onPoll`**: The polling implementation. An adapter function to be invoked at specified intervals

Sending 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', 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
 };
}
```

The `submitNotification` function takes the recipient user identifier and the notification text as parameters.

```
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` method creates and returns a default notification JSON block for the supplied values.

```
var badgeDigit = 1;<br />
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`**: The `custom` argument, or payload, is a JSON object that is transferred to the application and can contain custom properties.

The `WL.Server.notifyAllDevices` 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 a 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. It takes the following arguments:

```
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.

Notification API - Interactive notification

Note: Supported by iOS 8 only.

With interactive notification, when a notification arrives, users cantake actions without opening the application. When an interactive notification arrives, the device shows action buttons along with the notification message. Currently, interactive notifications are supported in devices that run iOS 8 and later. If an interactive notification is sent to iOS devices with version earlier than 8, the notification actions are not displayed.

> For more information, see the topic about interactive notifications, in the user documentation.

Notification API - Tag-based notification

By using this notification type, you can enable messages to be sent and received depending on tags.
Tags represent topics of interest to the user and provide the ability to receive notifications according to the chosen interest.

Tags are defined in the `application-descriptor.xml` file:

```
<tags>
  <tag>
    <name>PushTag1</name>
    <description>About pushTag1</description
>
  </tag>
  <tag>
    <name>PushTag2</name>
    <description>About pushTag2</description
>
  </tag>
</tags>
```

Client-side API methods

- `WL.Client.Push.subscribeTag(tagName,options)`: Subscribes the device to the specified tag name
- `WL.Client.Push.unsubscribeTag(tagName,options)`: Unsubscribes the device from the specified tag name
- `WL.Client.Push.isPushSupported()`: Returns `true` if push notifications are supported by the platform, or `false` otherwise
- `WL.Client.Push.isTagSubscribed(tagName)` - Returns whether the device is subscribed to a specified tag name

> For more information about tag-based notification, see the topic about tag-based notification, in the user documentation.

Notification API - Broadcast notification

Broadcast notifications are enabled by default for any push-enabled MobileFirst application. A subscription to a reserved tag, `Push.ALL`, is created for every device.
You can disable broadcast notification by unsubscribing from the reserved tag `Push.ALL`.

> For more information about broadcast notification, see the topic about broadcast notification, in the user documentation.

Common APIs for tag-based and broadcast notifications

Client-side API

`WL.Client.Push.onMessage (props, payload)`
props: A JSON block that contains the notifications properties of the platform.
payload: A JSON block that contains other data that is sent from the MobileFirst Server. The JSON block also contains the tag name for tag and broadcast notification. The tag name appears in the element. For broadcast notification, the default tag name is `Push.ALL`.

```
WL.Client.Push.onMessage = function (props, payload) {
    alert("Provider notification data: " + Object.toJSON(props));
    alert("Application notification data: " + Object.toJSON(payload));
}
```

A callback function that is invoked when a push notification is received by the device.
Set this function at a global JavaScript level. The tag name `Push.ALL` is sent back in the `payload` parameter.

Server-side API

`WL.Server.sendMessage(applicationId,notificationOptions)`
applicationId: (mandatory) The name of the MobileFirst application.
notificationOptions: (mandatory) A JSON block containing message properties.

This method submits a notification, based on the specified target parameters.

> For a full list of message properties, see the user documentation.

Project setup and guidelines

The way you set up push notifications depends on the platform.

Android

To send push notifications to Android devices, you must use the Google Cloud Messaging (GCM).
Use valid Gmail account to register to the GCM service.

For more information about how to get a GCM Project Number and API key (to be used later in the MobileFirst project), see the Cloud Messaging (http://developer.android.com/guide/google/gcm/gs.html) page at `http://developer.android.com/guide/google/gcm/gs.html`.

Notes:

- When creating an API key, make sure that the type is **server key**.
- Android OS 2.3.5 devices must be synchronized with a Gmail account.
- Android OS 4.x devices do not impose Gmail account synchronization.

iOS

To send push notifications to iOS devices, you must use the Apple Push Notifications Service (APNS).
As a developer, you must be a registered Apple iOS Developer to obtain an APNS certificate for your application.
Note: APNS certificates must have a non-blank password.

- In the development phase, the sandbox certificate file should be `apns-certificate-sandbox.p12` and be placed in the environment root folder or in the application root folder. The environment root folder takes the highest priority.
- In production, the production certificate file should be `apns-certificate-production.p12` and be placed in the environment root folder or in the application root folder. The environment root folder takes the highest priority.

When the hybrid application has both iPhone and iPad environments, it requires separate certificates for each environment. In that case, place those certificates in the corresponding environment folders.

Windows Phone 8

To send push notifications to Windows Phone 8 devices, you must use the Microsoft Push Notifications Service (MPNS).

Non-authenticated push notification does not require any setup from the developer.
Authenticated push notification requires a Windows Phone Dev Center account.

To use authenticated push, you must use a certificate that is issued by a Microsoft-trusted root certificate authority.
Note: In production mode, use authenticated push notification to ensure that data security is not compromised.

To send push notifications, the following servers must be accessible from a MobileFirst Server:
iOS:
Sandbox servers:
gateway.sandbox.push.apple.com:2195
feedback.sandbox.push.apple.com:2196

Production servers:
gateway.push.apple.com:2195
Feedback.push.apple.com:2196

1-courier.push.apple.com 5223

Android

If your organization has a firewall that restricts the traffic to or from the Internet, you must go through the following steps:

1. Configure the firewall to allow connectivity with GCM so that your GCM client apps can receive messages. The ports to open are 5228, 5229, and 5230. GCM typically uses only 5228, but it sometimes uses 5229 and 5230. GCM does not provide specific IP, so you must allow your firewall to accept outgoing connections to all IP addresses that are contained in the IP blocks listed in Google ASN of 15169. For more information, see Implementing an HTTP Connection Server (https://developers.google.com/cloud-messaging/http).
2. Ensure that your firewall accepts outgoing connections from MobileFirst Server to `android.googleapis.com` on port 443.

Windows Phone 8

No specific port needs to be open in your server configuration. MPNS uses regular http or https requests.

Setup - Project configuration

To set up push notifications in an application, add the following lines to the `application-descriptor.xml` file.
You can also modify these settings with the Application Descriptor Editor in Design mode.

Android

Use the values that you previously created in the GCM website:

- Place the **API Key** value instead of **GCM_Key**.
- Place the **Project Number** value instead of **senderId**.

```
<android securityTest="PushApplication-strong-mobile-securityTest" version="1.0">
    <pushSender key="GCM_key" senderId="GCM_ID"/>
```

iOS

Place the Apple APNS certificate file at the root of the application folder or at the root of the environment folder.
Replace `certificate password` with your actual certificate password.
Replace `com.PushNotifications` with the `bundleId` of your application. Consult the Apple documentation about how to create `bundleId` for Xcode projects.

```
<iphone bundleId="com.PushNotifications" version="1.0">
  <pushSender password=""/>
```

Windows Phone 8

To set up non-authenticated push:

```
<windowsPhone8 version="1.0">
  <uuid>e446f9d1-8d04-4198-be53-9fb44ae47548</uuid
>
  <pushSender/>
```

To set up authenticated push:

```
<windowsPhone8 version="1.0">
  <uuid>e446f9d1-8d04-4198-be53-9fb44ae47548</uuid>
  <pushSender><br />
  <authenticatedPush serviceName=" ... " keyAlias=" ... " keyAliasPassword=" ... "/></pushSender>
```

> For more information about using the certificate file, see the topic about setting up push notification for Windows Phone 8, in the user documentation.

Back-end emulator

The sample project for this tutorial is bundled with a back-end emulator. You can use it to simulate push notification submissions by a back-end system.
The source for the emulator can be found in the associated sample.

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
```

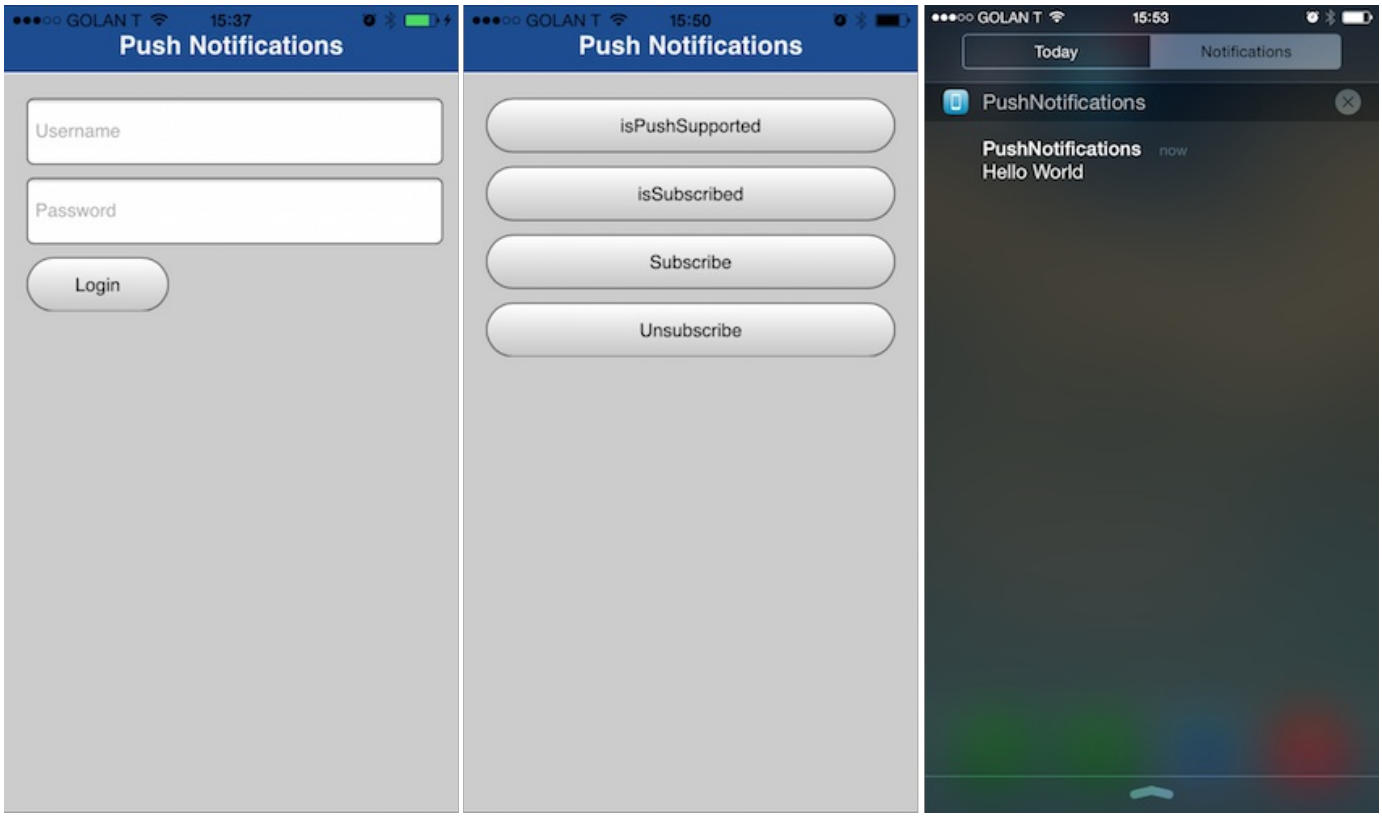
The back-end emulator tries to connect to a MobileFirst Server and call a `submitNotification()` adapter procedure.
It outputs the invocation result to a command prompt console.

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/v630/PushNotificationsHybridProject.zip) the Studio project.