MobileFirst Platform {dev}

Event Source Notifications in Hybrid Applications

Relevant to:
Hybrid

Overview

Prerequisite: Make sure to read the Push Notifications in Hybrid 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 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
- Displaying a received messaged

Agenda

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',
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 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 (available only in iOS and Windows Phone).
- **`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

- `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 application 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 is also bundled.

To use the back-end emulator: open the PushBackendEmulator folder in a command prompt/terminal and 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
```

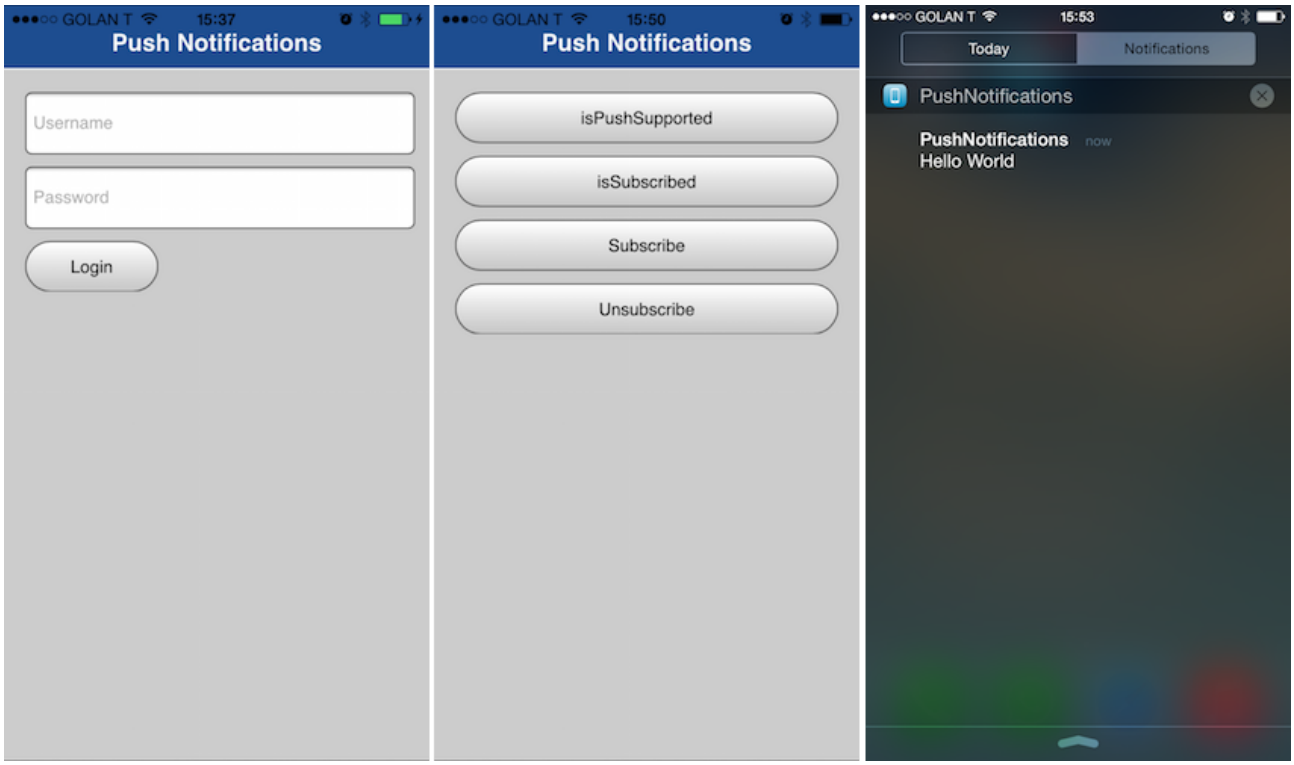
The back-end emulator tries to connect to a MobileFirst Server and call a `submitNotification()` adapter procedure. It outputs the invocation result to a command prompt console.

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 the MobileFirst project.