SMS Notification

Overview

This tutorial describes how to use the API for SMS notification in hybrid applications.

The following topics are covered:

- What is SMS notification?
- Supported devices
- Notification architecture
- Subscription management
- SMS Notification API
- Using a Subscribe SMS servlet
- Setup
- Back-end emulator
- Sample application

What is SMS notification?

SMS notification is the ability of a mobile device to receive notifications as SMS messages that are pushed from a server. Notifications are received regardless of whether the application is running.

Supported devices

IBM® MobileFirst Platform Foundation® supports SMS notifications on iOS, Android, Windows Phone, and BlackBerry devices that support SMS functions.

Notification architecture

Terminology

- Event source: A notification channel to which mobile applications can subscribe.
  - An event source is defined within a MobileFirst adapter.
- User ID: A unique identifier for a MobileFirst user
  - A user ID is obtained through authentication or through another unique identifier such as a persistent cookie.
- Application ID: MobileFirst application ID
  - This ID identifies a specific MobileFirst application on the mobile market.

Subscription

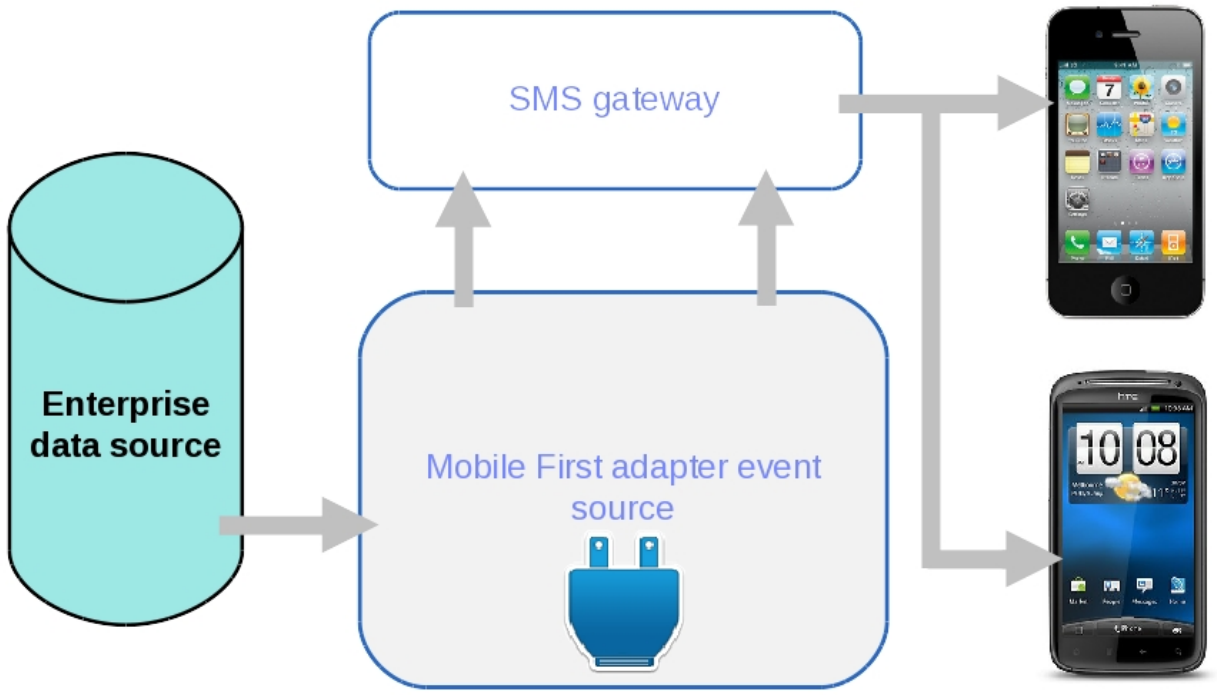
To start receiving SMS notifications, an application must first subscribe to an SMS notification event source. An event source is declared in the MobileFirst adapter that is used by the application for SMS notification services. To subscribe to SMS notifications, the user supplies a mobile phone number and approves the notification subscription. A subscription request is sent to the MobileFirst Server upon receipt of the user approval.

Sending notifications

MobileFirst provides a unified push notification API. With the adapter API, you can:

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

With the application API, you can subscribe to, and unsubscribe from, push notification event sources. To send a notification, you must first retrieve it from the back-end system. 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 a preconfigured SMS gateway. You can add extra custom logic to the adapter to preprocess notifications. The SMS gateway receives the notification and sends it to a device.



The process rolls out as follows:

1. Notifications are retrieved by the event source declared in the MobileFirst adapter, by either poll or push from the back-end system.
2. A MobileFirst adapter processes the notification and sends it to an SMS gateway.
3. The SMS gateway sends a push notification to the device.
4. The device processes the push notification as an SMS message.

Subscription management

User subscription

- User subscription

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

- Creation
  The user subscription for an event source is created when the user subscribes to that event source for the first time from any device.
- Deletion
  A user subscription is deleted when the user unsubscribes from that event source from all of their devices.
- Notification
  While the user subscription exists, the MobileFirst Server instance can produce notifications for the subscribed user. These notifications can be delivered by the adapter code to some or all of the subscribed devices.

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.subscribeSMS` API. The mobile phone number is an input parameter. The device subscription is deleted either when an application calls the `WL.Client.Push.unsubscribeSMS` API, or when the administrator unsubscribes the user through the MobileFirst Operations Console.

SMS notification API

Server side

1. Start by creating an event source:
   Declare a notification event source in the adapter JavaScript code at a global level (outside any JavaScript function).

```
WL.Server.createEventSource({
      name: 'SMSEventSource',
      onDeviceSubscribe: 'onDeviceSubscribeCallback',
      onDeviceUnsubscribe: 'onDeviceUnsubscribeCallback'
,
      securityTest:'SMSRealm-mobile-securityTest'
      poll:{
            interval: 3,
            onPoll: getNotificationFromBackend
      }
});
```

   - `name` – The name by which the event source is referenced.
   - `onDeviceSubscribe` – The adapter function that is called when the user subscription request is received.
   - `onDeviceUnsubscribe` – The adapter function that is called when the user unsubscription request is received.
   - `securityTest` – The security test from the `authenticationConfig.xml` file that is used

to protect the event source.

- `poll` – The method that is used for notification retrieval, with the following required parameters:
  - `Interval`: The polling interval in seconds
  - `onPoll`: The polling implementation: an adapter function to be invoked at specified intervals
- Send a notification:

```
function sendSMS(userId, smsText){
	var userSubscription = WL.Server.getUserNotificationSubscription('SMSAdapter.S
MSEventSource', userId);
	if (userSubscription==null){
		return { result: "No subscription found for user :: " + userId };<br />
	}
	var badgeDigit = 1;
	var notification = WL.Server.createDefaultNotification(smsText, badgeDigit, {});
	WL.Logger.debug("sendSMS >> userId :: " + userId + ", text :: " + smsText);
	WL.Server.notifyAllDevices(userSubscription, notification);
	return {
		result: "Notification sent to user :: " + userId
	};
}
```

Notifications can be either polled from, or pushed by, the back-end system. In this example, a `sendSMS()` adapter function is called by a back-end system as an external API to send notifications.

1. Retrieve the active user and use it to get the user subscription data.
2. Obtain notification data: The `sendSMS()` method takes the user ID to send the notification to, and the SMS text as arguments.
   These arguments are provided by the back-end system that calls this function.
   A user subscription object contains the information about all the user subscriptions. Each user subscription can have several device subscriptions. If the user has no subscriptions for the specified event source, a null object is returned.

   By using the `getDeviceSubscriptionsAPI` method, you can obtain separate subscription data for each user device.
3. Send notification to the user devices by using the following APIs:
   - `WL.Server.notifyAllDevices(userSubscription, options)`: To send notification to all the subscribed user devices.
   - `WL.Server.notifyDevice(userSubscription, device, options)`: To send notification to a specific device for a specific user subscription.
   - `WL.Server.notifyDeviceSubscription(deviceSubscription, options)`: To send notification to a specific device.

Client side

Prerequisite: Before subscription to, or unsubscription from, event sources, authentication is mandatory.

The `WL.Client.Push.subscribeSMS(alias, adapterName, eventSource, phoneNumber, options)` method takes the following parameters:

- alias - An alias to identify the subscription
- adapterName - An adapter name where the event source is defined
- eventSource - An event source name to which the client is subscribing
- phoneNumber - A user mobile phone number where notifications are sent
- options - Optional. A standard options object

The `WL.Client.Push.unsubscribeSMS(alias, options)` method takes the following parameters:

- alias - The same alias defined in `WL.Client.Push.subscribeSMS()`
- options - Optional. A standard options object.

Callbacks receive a response object if one is required.

Additional client side API:

- `WL.Client.Push.isPushSMSSupported()`: Returns `true` if SMS notification is supported by the platform, `false` otherwise.
- `WL.Client.Push.isSMSSubscribed(alias)`: Returns `true` if the currently logged-in user is subscribed to a specified event source alias, `false` otherwise.

Using a Subscribe SMS servlet

Subscription to SMS notifications is possible through a Subscribe SMS servlet. In this case, subscribing to SMS notifications is done through HTTP `GET` calls to the servlet, with no application on the device.

The Subscribe SMS servlet supports both subscribe and unsubscribe options. Before the call to the servlet, make sure that an application and event source within an adapter are deployed to MobileFirst Server.

Subscribe URL

`http://host:port/context/subscribeSMS?alias=aliasname&appId=appid&eventSource=eventsourcename&username=username&phoneNumber=number`

Unsubscribe URL

`http://host:port/context/subscribeSMS?option=unsubscribe&eventSource=eventsourcename&userName=username&phoneNumber=phonenumber`

URL parameter	Usage
option	Optional. The default value is `subscribe`. For any non-blank string other than `subscribe`, the `unsubscribe` action is performed.
event source	Mandatory. *`AdapterName.EventSourceName`*
alias	Optional. A short ID defining the event source name.
phone number	Mandatory. The number to which the SMS notifications are sent.

user name	Optional. If no value is provided, the phone number is used as the user name.
app ID	Mandatory. The ID of the application that contains SMS gateway definition. For example: `SMSPushApp-android-1.0`.

Subscriptions that are made through the Subscribe SMS servlet are independent of subscriptions that are created from a device. Unsubscribing by using SMS servlet does not remove subscriptions that are made from the device.

Security

By default, the Subscribe SMS servlet is protected as a static resource. By default, the `authenticationConfig.xml` file is configured to reject all requests to the Subscribe SMS servlet by using a rejecting login module. To allow access to the Subscribe SMS servlet, an administrator must modify the `authenticationConfig.xml` file with appropriate authenticator and login modules.

By default, the `authenticationConfig.xml` file is configured to reject all requests to the Subscribe SMS servlet:

```
<resource id="subscribeServlet" securityTest="SubscribeServlet">
   <urlPatterns>/subscribeSMS*</urlPatterns>
</resource>
<customSecurityTest name="SubscribeServlet">
   <test realm="SubscribeServlet" isInternalUserID="true"/>
</customSecurityTest>
<realm name="SubscribeServlet" loginModule="rejectAll">
   <className>com.worklight.core.auth.ext.HeaderAuthenticator</className>
</realm>
<loginModule name="rejectAll">
   <className>com.worklight.core.auth.ext.RejectingLoginModule</className
>
</loginModule>
```

An administrator can change the `authenticationConfig.xml` file to allow access to the Subscribe SMS servlet:

```
<resource id="subscribeServlet" securityTest="SubscribeServlet">
   <urlPatterns>/subscribeSMS*</urlPatterns>
</resource>
<customSecurityTest name="SubscribeServlet">
   <test realm="SMSRealm" isInternalUserID="true"/>
</customSecurityTest>
<realm name="SMSRealm" loginModule="smsHeader">
   <className>com.worklight.core.auth.ext.HeaderAuthenticator</className
>
</realm>
<loginModule name="smsHeader">
   <className>com.worklight.core.auth.ext.HeaderLoginModule</className>
</loginModule>
```

Setup

SMSConfig.xml

To send SMS notifications, define in the `SMSConfig.xml` file the third-party gateway services that are supported by MobileFirst Server.

```
<gateway id="myGateway" hostname="yourhostname.com" port="80" programName="backendProgra
m" toParamName="to" textParamName="text">
<parameter encode="false" name="param1name" value="param1value"/>
<parameter encode="false" name="param2name" value="param2value"/>
```

application-descriptor.xml

For an application to use SMS notifications, specify the SMS gateway to use by adding an element to the `application-descriptor.xml` file.

```
<smsGateway id=“myGateway”/>
```

The gateway ID in the `SMSConfig.xml` file must correspond to the SMS gateway ID in the `application-descriptor.xml` file.

Back-end emulator

The sample for this tutorial comes with a back-end system emulator, which you can use to simulate SMS notification submission by a back-end system.
To run the back-end system emulator:

1. Open a command prompt.
2. Go to the `SMSBackendEmulator` folder of the sample project.
3. Enter the following command to run the supplied `SMSBackendEmulator.jar` file:
   `java –jar SMSBackendEmulator.jar userId notificationText serverPort`
   The `userID` parameter is the user name that you used to log in to the sample application.

The back-end system emulator tries to connect to a MobileFirst Server instance and calls the `sendSMS()` adapter procedure. Then, it produces the invocation result to a command prompt console.

Sample application

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