SMS Notifications

Overview

In this training module, the developer will learn about the API and usage of SMS notifications with respect to Hybrid applications.

Topics covered, are:

- What are SMS notifications?
- Device support
- Notification architecture
- Subscription management
- SMS Notification API
- Subscribe SMS servlet
- Setup

What are SMS notifications?

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.

Device Support

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

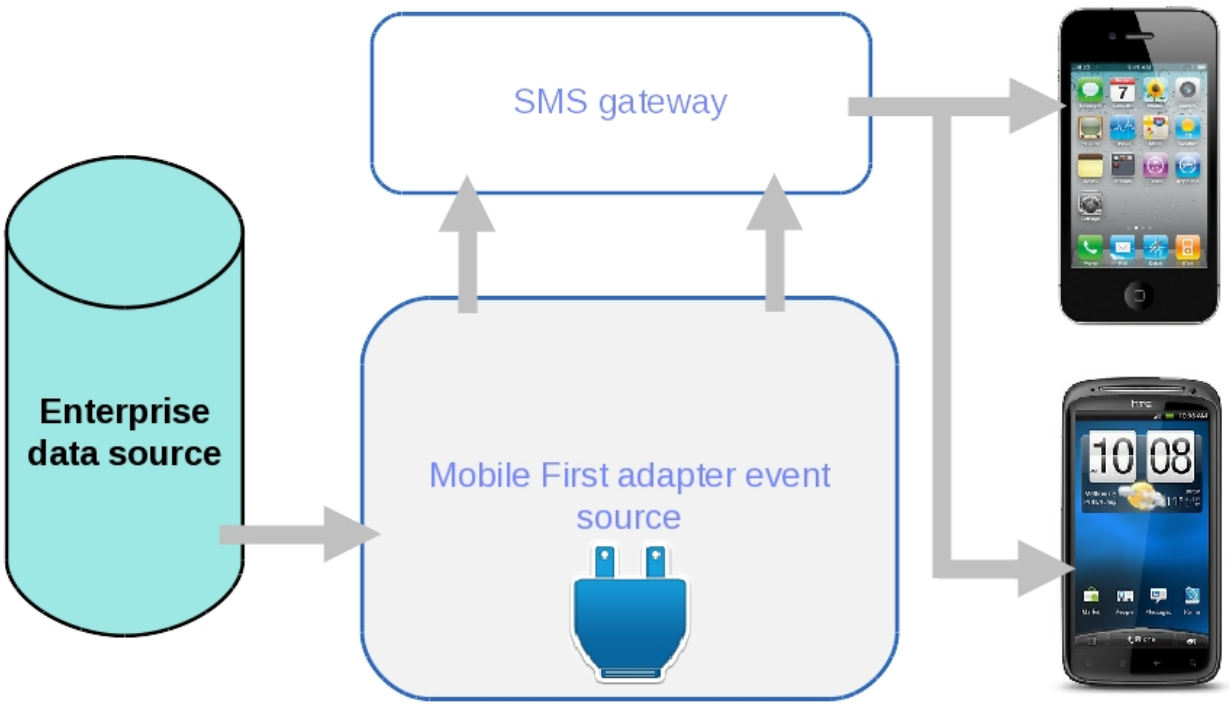
To start receiving SMS notifications, an application must first subscribe to an SMS notification event source. An event source is declared in the MobileFirst adapter that is used by the application for SMS notification services. To subscribe to SMS notifications, the user supplies a mobile phone number 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. Extra custom logic can be added to the adapter to preprocess notifications. The SMS gateway receives the notification and sends it to a device.



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

SMS Notification API

Server side

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–name by which the event source is referenced
- onDeviceSubscribe–adapter function that is called when the user subscription request is received
- onDeviceUnsubscribe–adapter function that is called when the user unsubscription request is received
- securityTest–security test from authenticationConfig.xml that is used to protect the event source
- poll–method that is used for notification retrieval, with the following required parameters:
  - Interval: polling interval in seconds
  - onPoll: polling implementation: an adapter function to be invoked at specified intervals

Send a notification:

```
function sendSMS(userId, smsText){
	var userSubscription = WL.Server.getUserNotificationSubscription('SMSAdapter.SMSEventSource'
, userId);
	if (userSubscription==null){
		return { result: "No subscription found for user :: " + userId };
	}
	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.
Retrieve the active user and use it to get the user subscription data.
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`, you can obtain separate subscription data for each user device. Send notification to the user devices: `WL.Server.notifyAllDevices` API sends notification to all the subscribed user devices.

To send notifications, the following APIs are used:
`WL.Server.notifyAllDevices(userSubscription, options)`: to send notification to all user devices (see previous slide)
`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

Subscribing or un-subscribing to eventsources can be achieved only upon authenticating first.

`WL.Client.Push.subscribeSMS()` has the following parameters:

- An alias to identify the subscription
- An adapter name where the event source is defined
- An event source name to which the client is subscribing
- A user mobile phone number where notifications are sent
- Optional onSuccesscallback
- Optional onFailurecallback

`WL.Client.Push.unsubscribeSMS` has the following parameters:

- An alias -the same as that defined in WL.Client.Push.subscribeSMS
- Optional onSuccesscallback
- Optional onFailurecallback

Callbacks receive a response object if one is required.

Additional client side API:
`WL.Client.Push.isPushSMSSupported()`: returns *true* if SMS notifications are 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.

Subscribe SMS servlet

Subscribing to SMS notifications can be achieved using a Subscribe SMS servlet. Subscribing to SMS notifications can be achieved without an application on the device, by making HTTP GET calls to the servlet.

The Subscribe SMS servlet supports both subscribe and unsubscribe options. An application and event source within an adapter must be deployed to the MobileFirst Server before making the servlet call.

Subscribe URL:

http:///:///subscribeSMS?alias=&appId=&eventSource=&username=&phoneNumber=

Unsubscribe URL:

http:///:///subscribeSMS?option=unsubscribe&eventSource=&userName=&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 exampe: SMSPushApp-android-1.0.

Subscriptions that are made by using 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 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, authenticationConfig.xml 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 authenticationConfig.xml 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 the MobileFirst Server.

```
<gateway id="myGateway" hostname="yourhostname.com" port="80" programName="backendProgram
" 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 training module comes with a back-end system emulator that 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 SMSBackendEmulatorfolder of the sample project.
3. Enter the following command to run the supplied SMSBackendEmulator.jar file:
   java –jar SMSBackendEmulator.jar
   where 'userID' is the user name that you used to log into the sample application.

The back-end system emulator tries to connect to a MobileFirst Server and call the sendSMS()adapter procedure. It then produces the invocation result to a command prompt console.