Nokia notifications on the Asha software platform

This article explains how to implement and use the Nokia Notifications API (for Java) on the Nokia Asha software platform.

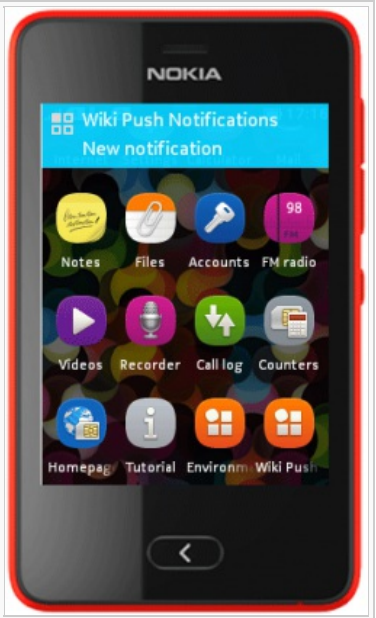
16 Jun 2013

Introduction

The Nokia Notifications API (introduced with the Nokia Asha software platform 1.0) provides a mechanism to deliver app-specific push messages from a remote server to a Java app. Notifications bring several advantages over alternative approaches (for instance, periodical server polling) including the following:

- Message delivery when the app is not running
- Fast message delivery
- Reduced battery consumption

This article shows how to implement a complete system that allows the setup and delivery of Nokia notifications.



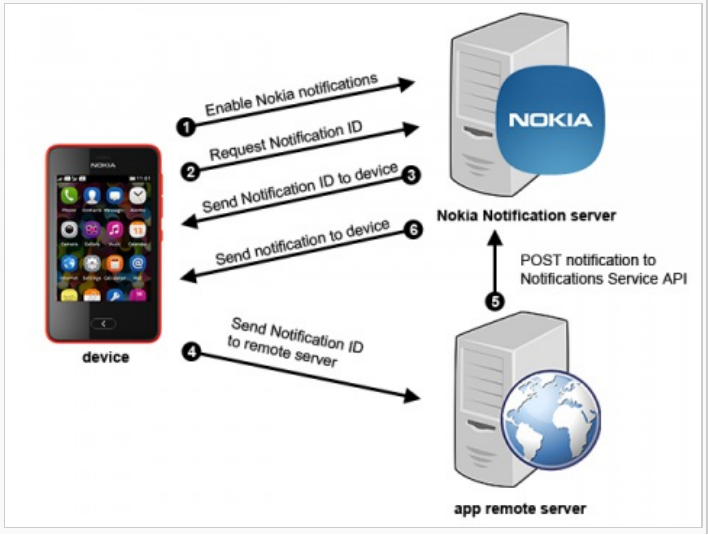
Notification from test app arriving in device

Notification system overview

The system includes:

- A Java app that uses the Nokia Notifications Client API to enable and handle Nokia notifications
- A remote Web server that pushes Nokia notifications to the Java app instance by using the Nokia Notifications Service API
- The Nokia Notification server, that communicates with both the Java app and the remote server

The following picture summarizes the interactions among the various elements:



Overview of the Nokia Notification Service

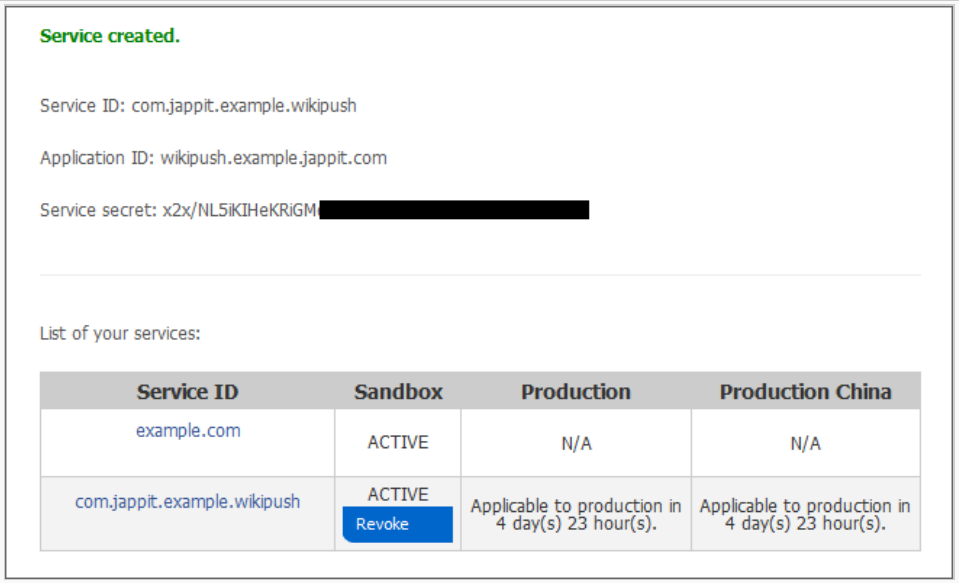
1. On **Step 1** the Java app takes care of enabling Nokia notifications
2. After a successful activation, on **Step 2**, the Java app requests the *Notification ID*, that is sent back to the app on **Step 3**
3. In order to be able to send notifications to the Java app, the remote server needs to receive and store its Notification ID, that is sent by the app on **Step 4**
4. When the remote server needs to send a notification to the Java app, it sends an *HTTP POST request* to the Nokia server, as shown by **Step 5**
5. **Step 6** finally illustrates how Nokia server takes care of actually delivering the notification to the Java app

Registering the notification service

Before actually integrating Nokia notifications in a Java app, it's necessary to register the specific notification service using the Notifications API Developer Console. This step is needed in order to get all the information needed to integrate the service in an app, and specifically:

- service ID
- application ID
- service secret

Note: Since the service ID must be unique across all apps and services, it is good practice to use the Reverse domain name notation.



Registering the notification service

Implementing Nokia notifications in a Java app

The Nokia Notifications Client API provides full support to quickly implement and integrate Nokia notifications in a Java app.

The typical workflow that a Java app follows in order to integrate Nokia notifications is the following:

1. **Open a session** (NotificationSession) that allows the Java app to communicate with the Nokia Notification Enabler
2. **Register the application** (via NotificationSession.registerApplication) to enable the delivery of Nokia notifications
3. **Receive the Notification ID** sent from the Notification Enabler: the Notification ID is an unique identifier that can be used by a remote server to send notifications to that specific device
4. **Send the Notification ID to a remote server**: typically, a remote Web server will receive and store the device's Notification ID in a database, together with other information that is relevant to the requested notifications (for instance, an user could desire to receive notifications only for a soccer team - in this case, the remote server should store the team information in order to filter the notifications that should be sent to the device)
5. **Handle incoming notifications** by implementing the NotificationSessionListener.messageReceived method: this method has a single argument of (NotificationMessage type, that contains all the information of the received notification

Enabling Nokia notifications

As seen in the previous section, enabling notifications basically requires the Java app to open a `NotificationSession` and call its registerApplication method.

Opening a notification session

A `NotificationSession` is opened by using the NotificationSessionFactory.openSession static method. This method requires four arguments:

1. the MIDlet instance that requires the notification service
2. the service ID (this argument is deprecated, but must still be specified)
3. the application ID associated with the notification service
4. an object implementing the NotificationSessionListener interface

```
try
{
```

```
 session = NotificationSessionFactory.openSession(midlet, "com.jappit.example.wikipush", "wikipush.example.jappit.com", listener);
}
catch(Exception e)
{
 // handle the raised exception
}
```

The `NotificationSessionListener` interface defines the methods that will be responsible for managing all the messages, state changes and information related to the notification session. Specifically, the following methods need to be implemented:

- stateChanged: this method receives all the state changes notifications from the Notification Enabler, so that the Java app can properly manage them. Specifically, a session can be OFFLINE, CONNECTING, or ONLINE.
- infoReceived: this method is used to receive the notification information, containing the Notification ID
- messageReceived: this method is called when a new Nokia notification is received, so that the Java app can properly handle that

Registering the app to receive notifications

Once a session is opened, it's possible to use its registerApplication method to enable notfications, so that new messages are actually delivered to the device.

```
try
{
 session.registerApplication();
}
catch(NotificationException e)
{
 // handle the raised exception
}
```

The registerApplication is an *asynchronous* method, that means that it immediately returns, and its actual result is asynchronously handled by the `NotificationSessionListener` instance defined in the previous section.

Specifically, after the `registerApplication()` call, the stateChanged method is called to notify about the registering process, and to inform the app if it has been correctly registered (with an NotificationState.STATE_ONLINE value) or if any error occurred during the registration process.

```
public void stateChanged(NotificationState state)
{
 switch(state.getSessionState())
 {
 case NotificationState.STATE_CONNECTING:
  // Java app is trying to connect to the Notification Enabler to enable notifications
  break;
 case NotificationState.STATE_OFFLINE:
  // Java app is offline and will not receive notifications
  break;
 case NotificationState.STATE_ONLINE:
  // Java app is online and enabled to receive notifications
  break;
 }

        int error = state.getSessionError();

 if(error != NotificationError.ERROR_NONE)
 {
  // handle the received error
 }
}
```

Note: Error handling is particularly important, since it allows the Java app to notify the user about the notification state, and eventually retry to register the app with the Notification Enabler.

Error checking can be performed by using the NotificationState.getSessionError method.

The Notification ID

After the Java app has been correctly registered, it is necessary to retrieve the Notification ID: that is "the unique identifier" needed by the remote server to send notifications to that specific Java app on that specific device.

Retrieving the Notification ID

The Java app can request the notification information by using the NotificationSession.getNotificationInformation() method.

```
try
{
 session.getNotificationInformation();
}
catch(NotificationException e)
{
 // handle the raised exception
}
```

Being an operation that requires network connectivity, this method is executed *asynchronously* and the result, once it is available, is passed as a NotificationInfo object to the NotificationSessionListener.infoReceived.

By using the NotificationInfo.getNotificationId method, it's then possible to retrieve the Notification ID.

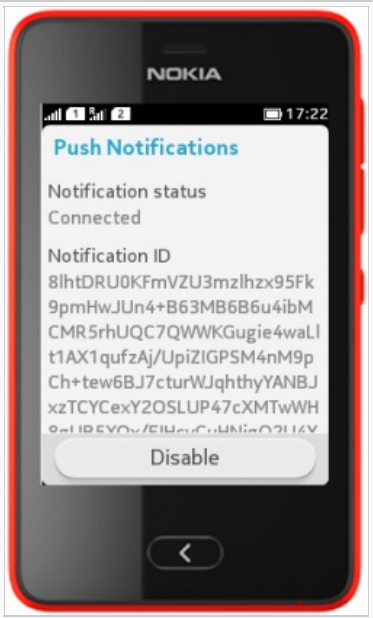
```
public void infoReceived(NotificationInfo info)
{
 String notificationId =  info.getNotificationId();

 if(notificationId.length() == 0)
```

```
 {
  // manage the case where an empty Notification ID is received
 }
 else
 {
  // manage the Notification ID, typically by storing it in persistent storage (e.g. RMS) and sending it to the notification remote
 }
}
```

Note: The Java app should always check if the returned Notification ID is an empty string: this case means that a network timeout occurred, and so the app must behave accordingly.

The following picture shows a screenshot of the sample app attached to this article, with the app displaying the `NotificationSession` state and the received Notification ID.



Test app displaying message

Sending the Notification ID to the remote server

Once retrieved, the Notification ID must be passed to the remote server that will be responsible for storing and then use that identifier to send notifications to that specific app instance.

Typically, this operation is performed by communicating with a Web server via an HTTP request, and this implies using, for instance, an HttpConnection or a SocketConnection object. Since implementing network connectivity logic is beyond the purpose of this article, its code is not illustrated here: anyway a fully functional sample is available in the attached source code ZIP file.

Handling incoming notifications

The previous sections explained how to enable notifications, and how to uniquely identify an app on a specific device to allow a remote server to send notifications to that specific instance.

What the Java app now must implement is the actual handling of incoming notifications. This operation is performed by implementing the messageReceived method of the `NotificationSessionListener` interface. The NotificationMessage instance, passed as argument to that method, can be used to retrieve all the notification information, including its title and payload.

Implementing the notification server

The Nokia Notification Server offers a Nokia Notifications Service API that allows to send notifications to Java apps. The Service API is a REST API, so allows interactions based on HTTP requests.

In order to send a notification to an app instance, a remote server sends an HTTP POST request to the Nokia Notifications Server endpoint *https://alpha.one.ovi.com/nnapi/1.0/nid/<NOTIFICATION_ID>/*, specifying the notification data (such as its content) in the request POST parameters.

The Nokia Notifications Server then sends back an HTTP response specifying if the notification was correctly sent (or queued for sending) or if any error occurred.

The HTTP requests must use SSL, together with Digest Access authentication. Depending on the programming language of choice, there are different SDKs and libraries that support such features.

The following code snippet shows a sample function that can be used to send a notification request to the Nokia Notifications Server by using the PHP programming language and curl. The function takes four parameters (service ID, service secret, notification ID and the notification message) and returns a boolean value indicating if the notification was correctly sent.

```
function sendNotification($serviceId, $serviceSecret, $notificationId, $message)
{
 // the alpha.one.ovi.com host must be used for testing in the sandbox environment
  $url = "https://alpha.one.ovi.com/nnapi/1.0/nid/" . urlencode($notificationId) . "/";
```

```
  // format the POST parameters
  $args = 'payload=' . urlencode($message) .
    '&ctype='.urlencode("text/plain");


  // initialize the curl session
  $session = curl_init($url);


  // set HTTP POST method
  curl_setopt ($session, CURLOPT_POST, true);

  // set the POST parameters
  curl_setopt ($session, CURLOPT_POSTFIELDS, $args);

  // return the HTTP request's response as a string instead of outputting it directly
  curl_setopt($session, CURLOPT_RETURNTRANSFER, true);

  // set the authentication method as requested by the Nokia Notification Server
  curl_setopt($session, CURLOPT_HTTPAUTH, CURLAUTH_DIGEST);

  // set the authentication data: service ID and service secret
  curl_setopt($session, CURLOPT_USERPWD, $serviceId . ':' . $serviceSecret);

  // execute the HTTP request
  $output = curl_exec($session);

  // get the response HTTP code
  $responseCode = curl_getinfo($session, CURLINFO_HTTP_CODE);

  curl_close($session);

  $success = ($responseCode == 201);

  return $success;
}
```

Note: The above function does not take into account important aspects that should be considered when deploying a solution in a production environment. Those aspects include:

- proper error management: it is important to check which error the Nokia Notifications Server sends back, in order to take the appropriate action (for instance, retry to send the notification, or remove the Notification ID for the successive notifications)
- performances: in real-world cases, it is often important how quickly a notification is sent and arrives to its receiver. The implementation of a notification system must take into account scalability, to be able to maintain good performances with increasing number of users.

Further considerations

Deactivating notifications

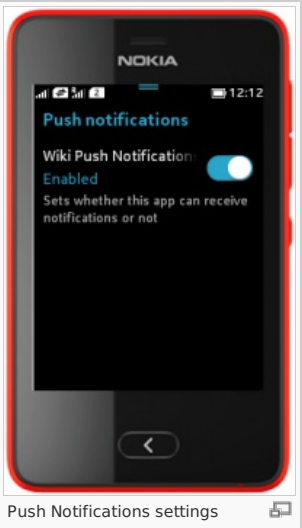
An app should always allow the user to deactivate notifications, once he/she doesn't desire those anymore. This can be done by using the NotificationSession.unregisterApplication method. As `registerApplication`, this is an asynchronous method, so it's necessary to use the NotificationSessionListener.stateChanged method to check its result.

When an user deactivates notification, unregistering the application, it is good practice to notify the remote server about this change, so that the server doesn't try to send further notifications to the user. Even if he/she wouldn't receive any more notifications, this step ensures that server resources are used in an optimal way, without wasting HTTP requests and bandwidth.

Delivery of notifications

Notifications are delivered when both the following conditions are met:

- the device has working network connectivity
- notifications for the Java app are enabled (notifications for each app can be enabled and disabled by the user within the device's settings screen, as shown by the following picture).



Push Notifications settings screen

If one of those conditions are not fulfilled, then the notifications are stored by the Nokia Notification Server, taking into account the following considerations.

- Only the last five notifications for a specific Notification ID are stored. When more notifications need to be stored, the oldest are removed and not

delivered.
- Notifications have a maximum expiration time of 14 days. After this time a notification is removed and not delivered.
- Each notification service has an hourly quota of notifications that can be sent. Over this quota no more notifications are delivered for the hourly period, and a 503 error is returned to Service API calls. The quota increases with the number of enabled Notification IDs.

Testing and production environments

The Nokia Notification Server allows to choose between sandbox and production environments. In order to properly setup and delivery notifications, both the Java app and the remote server must use the same environment.

- the Java app can be set to use the appropriate environment by using the Environment Selector MIDlet
- the remote server can select an environment by using the appropriate host:
  - *alpha.one.ovi.com* for the sandbox environment
  - *nnapi.ovi.com* for the production environment
  - *nnapi.ovi.com.cn* for the Chinese production environment

Filtering notifications

An user could desire to receive only notifications related to a specific topic or event. In this case, the topic/event identifiers must be associated with the Notification ID, so that the remote server is able to filter out the proper notifications for each user.

Notification system performances

As already stated, performances become an important matter with the increase of the number of app users. An effective solution can be represented by cloud servers, that allow to manage resources depending on the requests volume. Cloud solutions include:

- Windows Azure
- Amazon AWS
- Google App Engine

Sending notifications from the Developer Console

The Nokia Notifications API Developer Console offers a dedicated tool to test the delivery of Nokia notifications without the need to use a separate remote server. By using this tool, it's possible to send notifications with the desired payloads and parameters to the specified Notification ID.

Nokia Notifications API Developer Console

Summary

This article went through the main steps in implementing and integrating Nokia notifications in a Java app. More information is available on both the Nokia notifications developer's guide and the Nokia Notifications API JavaDocs pages.

The source code of the sample Java app illustrated in this article is available here: Media:WikiAshaNotifications.zip