How to launch a MIDlet remotely via SMS or locally via an Alarm with PushRegistry in Java ME

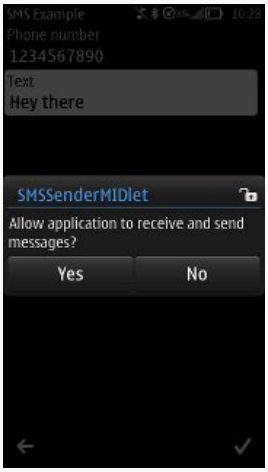
Warning: There is a bug in Nokia Asha 303, 305, 306 and 311 (only - to our knowledge) that means that the described approach will not work for a MIDlet that is registered *statically* (i.e. in the JAD) until it has been manually launched at least once. Dynamic registration within code works as expected.

15 Jul 2012

This code example demonstrates how to use the PushRegistry API in order to launch a MIDlet automatically when an SMS is received or a specified time period has elapsed.

Overview

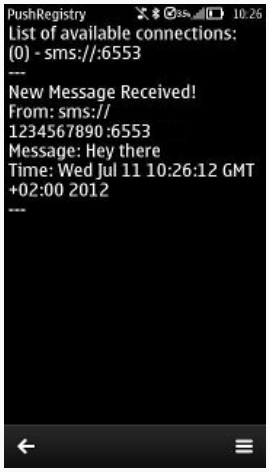
The first part of the code example shows how to send an SMS from device "A" to start an application in an idle device "B"



Device A sends an SMS

Receiving device B in idle

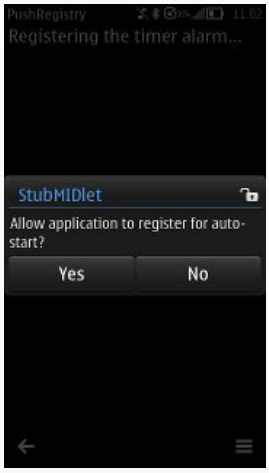
Device B is launching



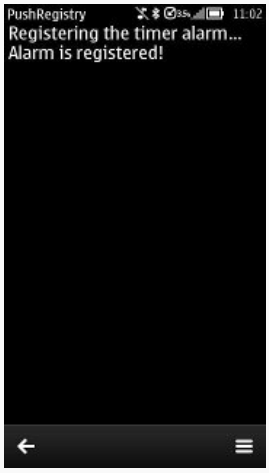
Device B has started and received notification

The second part shows how to start the idle device when a specified time period has elapsed.

Register device to start on alarm (from menu)



Acknowledge auto-start prompt



App notifies it is registered

App starting at specified time

(C) Copyright Nokia 2013 All rights reserved.

App has started (no notification!)

Registering to the push registry

MIDlets can be registered to the push registry by using the following techniques:

- Statically - through the `MIDlet-Push-<n>` attribute inside the application descriptor (the .jad file). This means the application is registered for listening to push notifications during the installation process.
- Dynamically (at runtime) - through the `registerConnection()` method of the `PushRegistry` class. This will register a single inbound connection, which is listened to by the MIDlet itself or by the application management software (AMS). When a notification arrives for a registered MIDlet, the AMS will start the MIDlet.
- Dynamically - through the `registerAlarm()` method of the `PushRegistry` class. This will register a timer alarm which will launch the MIDlet after a certain time period.

Note: The practical difference between a static and dynamic registration is the moment when the registration for listening to incoming push notifications takes place. In the static case, no action needs to be taken, other than installing the application. However, in the dynamic registration, the `registerAlarm()` and `registerConnection()` methods need to be called from within the MIDlet and added as actions under a UI Component, such as a Command.

In this example we demonstrate with sample code the differences between these techniques.

Note: Passing data to the application that is being launched is only feasible via a static or dynamic registration for listening to incoming push notifications from a certain port. Alarm notifications do not pass data.

Warning: Static and dynamic registration cannot co-exist for the same Connection URL. Static registration reserves the port the moment the application is installed and from that point on, dynamic registration on the same port will fail. In other words, If you install the static binaries Media:StubMIDletStaticBinaries.zip, you should not attempt to register dynamically the MIDlet from its Options Menu. Such an attempt will result to a warning that the requested registration at the given port has failed. In order to unregister the static registration, you need to uninstall the application. Also if you install the Media:SMSSenderMIDletBinaries.zip on the same device where you attempt either a static or dynamic registration to a certain port, the SMSSenderMIDlet will give you a failure warming, because the same port used for sending the SMS is reserved by the PushRegistry MIDlet. You need to use two separate devices, one where the SMSSenderMIDlet is installed, and another where the PushRegistry MIDlet is installed.

Static Registration for listening to incoming push notifications from a certain port

In order to register the MIDlet to the push registry statically, we need to define a `MIDlet-Push-<n>` property in the .jad file of the MIDlet:

```
MIDlet-Push-<n>: <ConnectionURL>, <MIDletClassName>, <AllowedSender>
```

where:

- `MIDlet-Push-<n>` is the property name that identifies push registration, and `<n>` is a number starting from 1. In this snippet, `MIDlet-Push-1` is used.
- `ConnectionURL` is a connection string that identifies the inbound endpoint to register. In this snippet, `"sms://:6553"` is used. It means that the MIDlet will be launched when an SMS is received in port 6553. This requires a custom SMS Sender MIDlet that sends an SMS to the mutually agreed port. In this example we re-use the SMSSenderMIDlet (Media:SMSSenderMIDletBinaries.zip) from the Listening asynchronously for incoming SMS messages in Java ME article.
- `MIDletClassName` is the fully qualified class name of the MIDlet to be activated when network activity in `ConnectionURL` is detected. In this case, `StubMIDlet` is used.
- `Allowed-Sender` is a filter used to restrict the servers that can activate `MIDletClassName`. Here, `*` is used. It means that this MIDlet can be activated by any sender.

Therefore, static registration of our StubMIDlet is implemented by adding the following line in the file descriptor:

```
MIDlet-Push-1: sms://:6553,StubMIDlet,*
```

When the application is launched, statically, the getPushRegistryConnections() method is called from within StartApp(). This action lists the registered connection and adds a message listener to MIDlet's instance.

```
String[] connections;
```


(C) Copyright Nokia 2010. All rights reserved.

```
    // Get a list of registered connections for this MIDlet
    connections = PushRegistry.listConnections(false);

    if (connections.length != 0) {
        printString("List of available connections: ");
        for (int i = 0; i < connections.length; i++) {
            try {
                MessageConnection mc =
                        (MessageConnection) Connector.open(connections[i]);
                printString("(" + i + ") - " + connections[i]);
                // Register this MIDlet to be notified when a message has
                // been received on the connection
                mc.setMessageListener(this);
```

The MIDlet implements the MessageListener

```
public class StubMIDlet
 extends MIDlet
 implements MessageListener, CommandListener {
```

so the method notifyIncomingMessage of the MessageListener interface is included in the source code:

```
public void notifyIncomingMessage(final MessageConnection conn) {
        // Because this method is called by the platform, it is good practice to
        // minimize the processing done here, on the system thread. Therefore,
        // let's create a new thread for reading the message.
        Thread smsThread = new Thread() {
            public void run() {
                try {
                    // Receive all incoming messages to the specified
                    // connection. The receive() method will block until there
                    // is a message available.
                    Message message = conn.receive();
                    if (message != null) {
                     //Get the text message and display it on the screen
                     TextMessage textMessage = (TextMessage)message;
                     mainForm.append("---\n");
                     mainForm.append("New Message Received!\n");
                     mainForm.append("From: " + textMessage.getAddress() + "\n");
                     mainForm.append("Message: " + textMessage.getPayloadText() + "\n");
                     mainForm.append("Time: " + textMessage.getTimestamp() + "\n");
                        mainForm.append("---\n");
                    }
                } catch (IOException ex) {
                    // Nothing to do here
                }
            }
        };
        smsThread.start();
 }
```

As it can be seen from above, the notifyIncomingMessage, receives the message from the registered MessageConnection, typecasts it to a TextMessage instance and retrieves the payload (text), address (sender's number) and timestamp in order to display them on the screen.

Dynamic registration for listening to incoming push notifications from a certain port

The only difference in the dynamic registration is that the MIDlet registers a PushRegistry connection when the appropriate command is selected by the end user:

```
public void commandAction(Command command, Displayable displayable) {
    if (command == EXIT_COMMAND) {
        // Exit the MIDlet
        notifyDestroyed();
    } else if (command == REGISTER_CONNECTION_COMMAND) {
        registerSMSConnection();
```

where registerSMSConnection() hides the call to the registerConnection(). The arguments for the dynamic registration, via the `PushRegistry.registerConnection()` method call, are the same as for static registration.

```
private void registerSMSConnection() {
    printString("Registering the connection...");

    try {
        // Register this MIDlet to be launched automatically when an SMS is
        // sent to the specified port
        PushRegistry.registerConnection(SMS_CONNECTION_URL, MIDLET_CLASS_NAME, ALLOWED_SENDER_FILTER);
        printString("Registration is complete!");
```

SMS_CONNECTION_URL and ALLOWED_SENDER_FILTER in this case, are defined as constants inside the source, instead of the application descriptor, as it would have happened in the case of static registration:

```
// Endpoint to register. This MIDlet will be launched when an SMS is
// received in the port 6553.
private static final String SMS_CONNECTION_URL = "sms://:6553";

// This MIDlet can be activated by any sender
private static final String ALLOWED_SENDER_FILTER = "*";
```

The variable MIDLET_CLASS_NAME is simply **StubMIDlet** in this case. It is instantiated in the constructor of our MIDlet as follows:

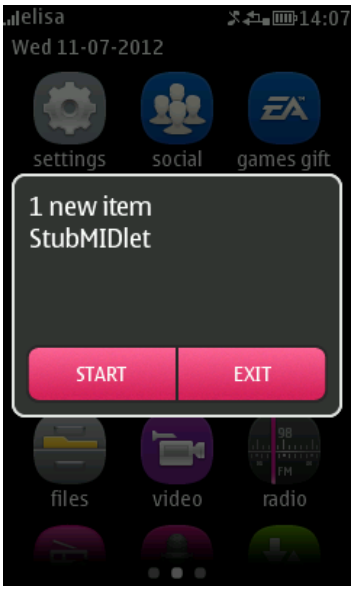
```
public StubMIDlet() {
    // The Class name is needed for the registration
    MIDLET_CLASS_NAME = this.getClass().getName();
```

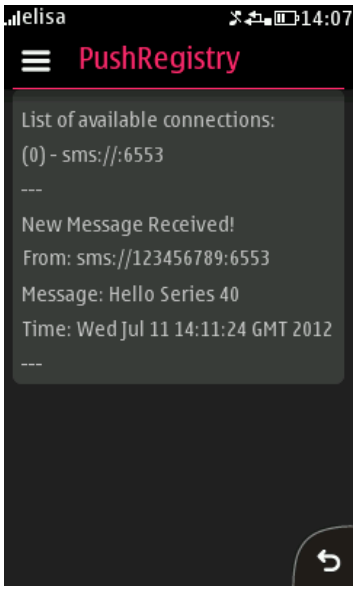
The MIDlet can be unregistered from the push registry at runtime by calling the `PushRegistry.unregisterConnection()` method.

(c) Copyright Nokia 2010. All rights reserved.

```
private void unregisterSMSConnection() {
    printString("Unregistering the connection...");
    PushRegistry.unregisterConnection(SMS_CONNECTION_URL);
    printString("Unregistration is complete!");
}
```

Each time a message is received to the connection URL specified above, an alert is displayed in the idle screen of Series 40 devices:





Note: The application management software asks for permission to launch the MIDlet. This is normal behavior for untrusted MIDlets. For more information, please refer to the Java Security Domains and Signing a MIDlet Suite articles.

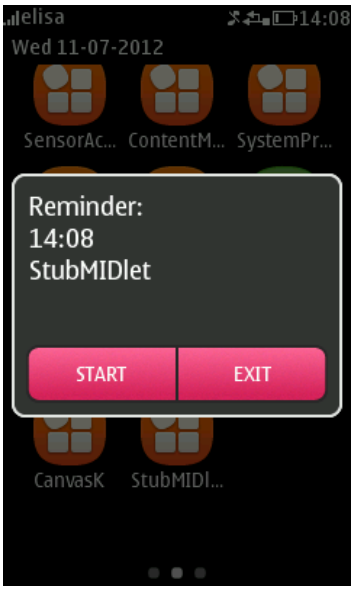
Dynamic registration for auto launch with an Alarm

In this example, we demonstrate how the MIDlet can also be registered to be launched automatically after 5 seconds through the "Register timer alarm" command. The registration is slightly different in this case, as a value in milliseconds needs to be passed to the registerAlarm() method as shown below:

```
private void registerTimerAlarm(long timePeriodToAutoStart) {
    // Set the launch time to current time + the specified period
    long timeToWakeUp = System.currentTimeMillis() + timePeriodToAutoStart;
    printString("Registering the timer alarm...");

    try {
        PushRegistry.registerAlarm(MIDLET_CLASS_NAME, timeToWakeUp);
        printString("Alarm is registered!");
    } catch (ClassNotFoundException ex) {
        printString(ex.getMessage());
        printString("Alarm registration failed.");
    } catch (ConnectionNotFoundException ex) {
        printString(ex.getMessage());
        printString("Alarm registration failed.");
    }
}
```

The alarm registration is displayed as a Reminder Alert in the latest Series 40 full touch devices:



(C) Copyright Nokia 2010. All rights reserved.

Note: According to the JSR-118 specifications for the PushRegistry, auto-launch can be triggered by a MIDlet that belongs to the same MIDlet Suite as the MIDlet that is being launched. This practically means that PushRegistry cannot be used for launching another, 3rd party MIDlet, that is installed on the device

Source code for StubMIDlet.java

```
import java.io.IOException;
import java.util.Vector;

import javax.microedition.io.ConnectionNotFoundException;
import javax.microedition.io.Connector;
import javax.microedition.io.PushRegistry;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.wireless.messaging.Message;
import javax.wireless.messaging.MessageConnection;
import javax.wireless.messaging.MessageListener;
import javax.wireless.messaging.TextMessage;


public class StubMIDlet
 extends MIDlet
 implements MessageListener, CommandListener {

 private final String MIDLET_CLASS_NAME;

 // Endpoint to register. This MIDlet will be launched when an SMS is
 // received in the port 6553.
 private static final String SMS_CONNECTION_URL = "sms://:6553";

 // This MIDlet can be activated by any sender
 private static final String ALLOWED_SENDER_FILTER = "*";

 private static final Command EXIT_COMMAND = new Command("Exit", Command.EXIT, 0);

 // Command that registers the MIDlet to launch automatically when an SMS is
 // received
 private static final Command REGISTER_CONNECTION_COMMAND = new Command("Register connection", Command.ITEM, 0);

 // Command that unregisters the MIDlet from auto launching by incoming SMS
 // message
 private static final Command UNREGISTER_CONNECTION_COMMAND = new Command("Unregister connection", Command.ITEM, 0);

 // Command that registers the MIDlet to launch automatically after a certain
 // time period
 private static final Command REGISTER_TIMER_ALARM_COMMAND = new Command("Register timer alarm", Command.ITEM, 0);

 // Available push connections for this MIDlet
 private Vector availableConnections = new Vector();

 private Form mainForm;

 public StubMIDlet() {
     // The Class name is needed for the registration
     MIDLET_CLASS_NAME = this.getClass().getName();
 }

 /**
  * Registers the connection for launching this MIDlet automatically by an
  * incoming SMS.
  */
 private void registerSMSConnection() {
     printString("Registering the connection...");

     try {
         // Register this MIDlet to be launched automatically when an SMS is
         // sent to the specified port
         PushRegistry.registerConnection(SMS_CONNECTION_URL, MIDLET_CLASS_NAME, ALLOWED_SENDER_FILTER);
         printString("Registration is complete!");

         // Update the inbound connections
         clearConnections();
         getPushRegistryConnections();
     } catch (ClassNotFoundException ex) {
         printString(ex.toString());
         printString("Registration failed.");
     } catch (IOException ex) {
         printString(ex.toString());
         printString("Registration failed.");
     }
 }

 /**
  * Unregisters the connection so that this MIDlet will not be launched when
  * an SMS is received.
  */
 private void unregisterSMSConnection() {
     printString("Unregistering the connection...");
     PushRegistry.unregisterConnection(SMS_CONNECTION_URL);
     printString("Unregistration is complete!");
 }

 /**
  * Registers this MIDlet to be launched automatically after a certain time
  * period.
  * @param timePeriodToAutoStart - period in milliseconds after which this
  * MIDlet will launch.
  */
 private void registerTimerAlarm(long timePeriodToAutoStart) {
     // Set the launch time to current time + the specified period
     long timeToWakeUp = System.currentTimeMillis() + timePeriodToAutoStart;
     printString("Registering the timer alarm...");

     try {
         PushRegistry.registerAlarm(MIDLET_CLASS_NAME, timeToWakeUp);
         printString("Alarm is registered!");
     } catch (ClassNotFoundException ex) {
         printString(ex.getMessage());
         printString("Alarm registration failed.");
     } catch (ConnectionNotFoundException ex) {
```


(c) Copyright Nokia 2010. All rights reserved.

```
        printString(ex.getMessage());
        printString("Alarm registration failed.");
    }
}

/**
 * Unregisters this MIDlet from all inbound connections and removes them
 * from the internal collection.
 */
private void clearConnections() {
    if (availableConnections != null) {
        while (!availableConnections.isEmpty()) {
            MessageConnection messageConnection =
                    (MessageConnection) availableConnections.firstElement();
            if (messageConnection != null) {
                try {
                    messageConnection.setMessageListener(null);
                    messageConnection.close();
                } catch (IOException ex) {
                    printString(ex.toString());
                }
            }
            availableConnections.removeElementAt(0);
        }
    }
}

/**
 * Establishes all available push connections.
 */
private void getPushRegistryConnections() {
    String[] connections;
    // Get a list of registered connections for this MIDlet
    connections = PushRegistry.listConnections(false);

    if (connections.length != 0) {
        printString("List of available connections: ");
        for (int i = 0; i < connections.length; i++) {
            try {
                MessageConnection mc =
                        (MessageConnection) Connector.open(connections[i]);
                printString("(" + i + ") - " + connections[i]);
                // Register this MIDlet to be notified when a message has
                // been received on the connection
                mc.setMessageListener(this);
                availableConnections.addElement(mc);
            } catch (SecurityException ex) {
                printString("Connection failed.");
                printString(ex.getMessage());
            } catch (IOException ex) {
                printString("Connection failed.");
                printString(ex.getMessage());
            }
        }
    }
}

/**
 * From CommandListener.
 * Called by the system to indicate that a command has been invoked on a
 * particular displayable.
 * @param command the command that was invoked
 * @param displayable the displayable where the command was invoked
 */
public void commandAction(Command command, Displayable displayable) {
    if (command == EXIT_COMMAND) {
        // Exit the MIDlet
        notifyDestroyed();
    } else if (command == REGISTER_CONNECTION_COMMAND) {
        registerSMSConnection();
    } else if (command == UNREGISTER_CONNECTION_COMMAND) {
        unregisterSMSConnection();
    } else if (command == REGISTER_TIMER_ALARM_COMMAND) {
        //registering timer alarm. Alarm will start MIDlet throught
        //30 seconds
        registerTimerAlarm(5000);
    }
}

/**
 * From MessageListener.
 */
public void notifyIncomingMessage(final MessageConnection conn) {
 // Because this method is called by the platform, it is good practice to
       // minimize the processing done here, on the system thread. Therefore,
       // let's create a new thread for reading the message.
       Thread smsThread = new Thread() {
           public void run() {
               try {
                   // Receive all incoming messages to the specified
                   // connection. The receive() method will block until there
                   // is a message available.
                   Message message = conn.receive();
                   if (message != null) {
                    //Get the text message and display it on the screen
                    TextMessage textMessage = (TextMessage)message;
                    mainForm.append("---\n");
                    mainForm.append("New Message Received!\n");
                    mainForm.append("From: " + textMessage.getAddress() + "\n");
                    mainForm.append("Message: " + textMessage.getPayloadText() + "\n");
                    mainForm.append("Time: " + textMessage.getTimestamp() + "\n");
                       mainForm.append("---\n");
                   }
               } catch (IOException ex) {
                   // Nothing to do here
               }
           }
       };
       smsThread.start();
}

protected void destroyApp(boolean arg0) throws MIDletStateChangeException {
 // TODO Auto-generated method stub

}

protected void pauseApp() {
 // TODO Auto-generated method stub

}
```


(C) Copyright Nokia 2013. All rights reserved.

```
  protected void startApp() throws MIDletStateChangeException {
   mainForm = new Form("PushRegistry");
   Display.getDisplay(this).setCurrent(mainForm);
   mainForm.addCommand(EXIT_COMMAND);
   mainForm.addCommand(REGISTER_CONNECTION_COMMAND);
   mainForm.addCommand(REGISTER_TIMER_ALARM_COMMAND);
   mainForm.addCommand(UNREGISTER_CONNECTION_COMMAND);
   mainForm.setCommandListener(this);
   getPushRegistryConnections();

  }

  private void printString(String message) {
   mainForm.append(message + "\n");
  }

 }
```

See also

- Listening asynchronously for incoming SMS messages in Java ME
- Listening synchronously for incoming SMS messages in Java ME
- The PushRegistry MIDP2.0 specifications
- How to send a message to a given port as SMS with Java ME

(C) Copyright Nokia 2010 All rights reserved.

(C) Copyright Nokia 2010 All rights reserved.