



<b>ID</b>	CS001422	<b>Creation date</b>	June 10, 2009
<b>Platform</b>	S60 3rd Edition FP2 S60 5th Edition S40 5th Edition LE	<b>Tested on devices</b>	Nokia N78 Nokia 5800 Nokia 2630
<b>Category</b>	Java ME	<b>Subcategory</b>	Base/System

**Keywords (APIs, classes, methods, functions):** javax.microedition.io.PushRegistry, javax.wireless.messaging.MessageConnection, javax.wireless.messaging.MessageListener, javax.wireless.messaging.TextMessage

## Overview

This snippet demonstrates how to use the PushRegistry API to launch a MIDlet automatically when a) an SMS is received or b) a specified time period has elapsed. MIDlets can be registered to the push registry by using the following techniques:

- Statically - through the MIDlet-Push-<n> property inside the application descriptor (the .jad file).
- 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.

This snippet describes all the above techniques.

To register the MIDlet to the push registry statically, 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://:6000" is used. It means that the MIDlet will be launched when an SMS is

received in port 6000.

- `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.

The arguments for the dynamic registration (the `PushRegistry.registerConnection()` method call) are the same as for static registration.

Note that in this example, when registering the connection dynamically, port 5000 is used for an incoming SMS.

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

**Note:** The application management software asks for permission to launch the MIDlet. This is normal behavior for untrusted MIDlets. For more information, refer to Signed MIDlets in the [Java ME Developer's Library](#).

## Source file: UsingPushRegistry.jad

```
MIDlet-1: StubMidlet, ,StubMIDlet
MIDlet-Jar-Size: 4713
MIDlet-Jar-URL: JavaMEStub.jar
MIDlet-Name: JavaMEExampleStub
MIDlet-Push-1: sms://:6000,StubMIDlet,*
MIDlet-Vendor: Vendor
MIDlet-Version: 1.1
MicroEdition-Configuration: CLDC-1.1
MicroEdition-Profile: MIDP-2.1
```

## Source file: UsingPushRegistry.jar

```
private final String MIDLET_CLASS_NAME;
// Endpoint to register. This MIDlet will be launched when an SMS is
// received in the port 5000.
private static final String SMS_CONNECTION_URL = "sms://:5000";
// 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
```

## CS001422 - Using PushRegistry in Java ME

```
// 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();

public StubMIDlet() {
    // ...

    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 {
```

## CS001422\_-\_Using\_PushRegistry\_in\_Java\_ME

```
        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.");
    }
}

/**
 * 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
        exit();
    } 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 through
        //30 seconds
        registerTimerAlarm(30000);
    } else if (command == EXECUTE_COMMAND) {
        // Execute the snippet
        executeSnippet();
    }
}

/**
 * From MessageListener.
 */
public void notifyIncomingMessage(MessageConnection conn) {
    // Nothing needs to be done here
}

```

## Postconditions

After the MIDlet is installed on the device, it will be registered for launching automatically when an SMS is received in port 6000.

After launching, the MIDlet can be dynamically registered/unregistered to be launched automatically when an SMS is received in port 5000. This is done with the "Register connection" and "Unregister connection" commands, respectively.

The MIDlet can also be registered to be launched automatically after 30 seconds through the "Register timer alarm" command.

## Supplementary material

This code snippet is part of the stub concept, which means that it has been patched on top of a template application in order to be more useful to developers. The version of the Java ME stub application used as a template in this snippet is v1.1.

- The patched, executable application that can be used to test the features described in this snippet is available for download at [Media:UsingPushRegistry.zip](#).

## CS001422\_-\_Using\_PushRegistry\_in\_Java\_ME

- You can view all the changes that are required to implement the above-mentioned features. The changes are provided in unified diff and colour-coded diff (HTML) formats in [Media:UsingPushRegistry.diff.zip](#).
- For general information on applying the patch, see [Using Diffs](#).
- For unpatched stub applications, see [Example stub](#).

## See also

[CS000976 - Sending a text SMS](#)