Push Registry in Java ME

Version 0.9, Draft

API GUIDE

COPYRIGHT

Trademarks and Service Marks

About This Document

This document describes the Push Registry API and its importance in Java ME application development followed by sample code snippet.

Scope

This document is intended for MIDP developers who want to develop mobile Java applications. This document emphasis on Push Registry API and its implementation, explaining the Java technology is out of the scope of this documentation.

Document History:

Date	Version	Comment
11/06/09	0.9	Draft

References:

1. Push Registry Article: http://developers.sun.com/mobility/midp/articles/pushreg/

Abbreviations:

MIDP	Mobile Information Device Profile
AMS	Application Management Software
SMS	Short Message Service
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
GCF	Generic Connection Framework

Table of Contents

Table of Figures

List of Tables

Introduction

A MIDlet is a MID Profile application which is implemented and controlled by the Application Management Software (AMS). MIDP 2.0 came up with many new features for developers to build innovative applications one of them being push registry. MIDP 2.0 PushRegistry feature provides a way for a MIDlet to respond to an inbound connection irrespective of whether the MIDlet is running or not. If the MIDlet is not open then MIDlet will be launched automatically to an incoming event.

PushRegistry can be used to respond to the following:

- Inbound wireless messaging connection such as SMS.
- Inbound network connection such as stream based TCP socket or packet based UDP datagram.
- Timer initiated MIDlet activation.

For Example:

PushRegistry can be used to notify the user when a work item has been created against his/her name, and user can respond to the work item as soon as possible.

Java ME's Push Registry easily pushes a message to a Java ME application and automatically launches the application.

You can also set events for appointments that have been scheduled or you can set timer-based activation to schedule your MIDlet.

Overview

javax.microedition.io.PushRegistry is the component of the AMS that exposes the Push API and keeps track of Push Registration.

Following steps show how PushRegistry works:

1. Connections like messaging (SMS) or Timer or network (socket, datagram) are needed to register a MIDlet application.
2. Push Registry maintains list of inbound connections associated with the application. Java ME application in the mobile device is registered for an event.
3. AMS monitors activity associated with the application.

4. When AMS detects an incoming connection associated with the MIDlet, AMS starts the MIDlet if it is not opened or delivers the response to the running MIDlet.

5. MIDlet now takes over responsibility for the connection and performs the steps necessary for handling the incoming connection.

Figure 1 shows the activation of PushRegistry on an incoming connection.

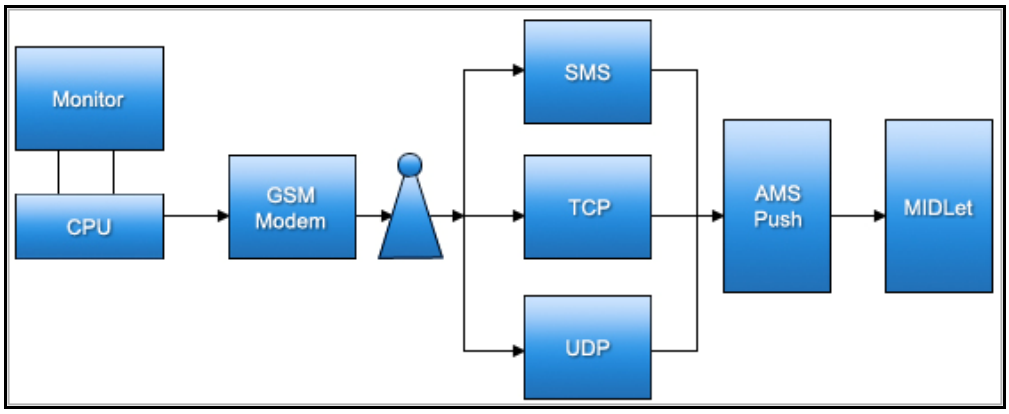

Figure 1: Activation of PushRegistry on an incoming connection

The Push Registry API

The Push Registry comes in a single class, *javax.microedition.io.PushRegistry* and is part of the Generic Connection Framework (GCF).

Table 1. describes the PushRegistry class.

Table 1: PushRegistry class

Class definition: public class PushRegistry extends Object	
Method	**Description**
getFilter()	Returns the filter <Allowed-Sender> for the specified push connection.
getMidlet()	Returns the MIDlet responsible for handling the specified Push connection.
listConnections()	Returns the list of registered Push connections for the MIDlet suite.
registerAlarm()	Registers a timer-based alarm to launch the MIDlet. Disables alarms if an argument of zero is passed.
registerConnection()	Registers a Push connection.
unregisterConnection()	Unregisters a Push connection.

Table 2 shows the exceptions that are to be handled by the application when using PushRegistry class methods.

Table 2: PushRegistry Exceptions

Exception	Thrown by	Description
ClassNotFoundException	registerConnection() registerAlarm()	The specified MIDlet class name cannot be found in the current MIDlet suite. In other words, the specified MIDlet class name was not defined (MIDlet-<n> attribute) in the descriptor file or the JAR file manifest, or the MIDlet argument is null.
ConnectionNotFoundException	registerConnection()	The platform does not support the specified connection type for Push inbound connections.
	registerAlarm()	The platform does not support alarm-based application launch.
	Connector.open()	The requested protocol does not exist, or the connection could not be made.
IllegalArgumentException	registerConnection()	The connection string is not valid, or the filter string is not valid.
	Connector.open()	One of the arguments is invalid.
IOException	registerConnection()	The connection is already registered, or there are insufficient resources to handle the registration.
	Connector.open()	A generic I/O error was encountered.
SecurityException	registerConnection()	The specified MIDlet does not have permission to register a connection.
	unregisterConnection()	The specified connection was registered by another MIDlet suite.
	registerAlarm()	The specified MIDlet does not have permission to register an alarm.
	Connector.open()	The MIDlet has no permission to use the requested protocol.

In MIDP 2.0, Security exception is a new feature. Here MIDP 2.0 applications must request permissions before using privileged operations such as network connections or the Push Registry. If you fail to request proper permissions, the platform may throw a SecurityException.

How to do Push Registration

Push Registration requires three important parts to be specified. They are:

1. ConnectionURL
2. MIDlet name
3. Filter

1. ConnectionURL

This is a connection string of the type that is passed to a *Connector.open()* method. ConnectionURL consists of protocol and port that describes the type of connection and the port on which the MIDlet will receive the inbound connection.
Example: sms://16001 or datagram://:16002 or socket://:16003.

2. MIDlet name:

MIDlet name is where you associate a MIDlet to the inbound connection declared by ConnectionURL. MIDlet name must be a fully qualified MIDlet class name and the MIDlet must be declared as part of the MIDlet suite in the same application descriptor.
Example: com.pr.PushMIDlet

3. Filter

Filter specifies the way to restrict the servers that can activate MIDlet. Two wildcards characters * and ? are supported and may be used to restrict the servers. The * matches any string including the empty string and ? matches any single character
Example 131.56.???.15 specify 3 characters at the 3rd position
and 131.56.*.15 specify 0 to 3 characters at the 3rd position

There are two ways to register the MIDlet application:

- Static Registration
- Dynamic Registration

Static Registration

During installation time, MIDlets are registered. This registration can be specified by listing *MIDlet-Push* attributes in the MIDlet suite's JAD or manifest file.

If the address you are trying to register is already bound, installation will fail. Uninstalling a MIDlet suite automatically unregisters the connection.

Given below are the parameters and their description, which should be added to make MIDlet static registration for a specific Port.

MIDlet-Push-<n>:
This is the attribute name of Push Registration. Multiple push registrations can be provided in a MIDlet suite. The numeric value for <n> starts from 1.
Example: MIDlet-Push-1, MIDlet-Push-2, MIDlet-Push-3 and so on.

ConnectionURL:
Enter the URL at which MIDlet has to be registered.
Example: sms: //:5000. This port number will be used by the server to trigger.

MIDletClassName:
Enter the name of the MIDlet, which should be triggered or activated. Example: com.sms.PushMidlet;

AllowedSender:
A designated filter that restricts the senders that is valid for launching the requested MIDlet
Wild card characters (*,?) can be also used here.
Example: 400.300.50.*, 300.300.20.1, *(Any IP address).

Example of added static Push Registration in JAD:

MIDlet-Push-1: sms://:5000, com.sms.PushMidlet, *;

Dynamic Registration

In dynamic registration, MIDlet is activated at runtime. Dynamic Registration of the port has to be implemented in the code using Push Registry class. You can register both dynamic connections and timer alarms at runtime, using Push Registry API.

Example:

In dynamic registration, add the following line of code:

```
PushRegistry.registerConnection("sms://:5000"," com.sms.PushMidlet ,*);
```

Unregistering Inbound Connection:
To unregister an inbound connection, use the *unregisterConnection()* method. Once the MIDlet registration is done, it will remain until the MIDlet suite is uninstalled from the device.

To unregister, dynamically add following code:

```
unregisterConnection("sms://5000");
```

The method returns true if it was successful and false if it fails to unregister the connection.

Registering Timer Alarms:

To register a MIDlet for alarm, dynamically use

PushRegistry.registerAlarm(String midletName, long alarmTime)

registeralarm(String midletName, long alarmTime) method requires fully qualified class name of the MIDlet to launch, and the time for the launch as parameters. Passing a time of zero disables the alarm.
Only one outstanding alarm per MIDlet is supported, and invoking this method overwrites any previously scheduled alarm.

Sample code on PushRegistry

```
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.midlet.*;
import javax.microedition.lcdui.*;
import javax.wireless.messaging.MessageConnection;
import javax.wireless.messaging.MessageListener;
import javax.wireless.messaging.TextMessage;
```

```
public class PushMidlet extends MIDlet implements CommandListener, MessageListener,
Runnable {

   Display display;
   Form form;

   /*MIDlet class name.*/
   private String midletName = this.getClass().getName();
   Command cmdExit = new Command("Exit", Command.EXIT, 1);
   /*Command for registering alarm*/
   Command cmdAlarm = new Command("Register Alarm", Command.ITEM, 0);
   /*Command for registering SMS Port*/
   Command cmdReg = new Command("Register SMS", Command.ITEM, 0);
   /*Command for unregistering SMS Port*/
   Command cmdUnreg = new Command("UNRegister SMS", Command.ITEM, 0);
   Vector allConn = new Vector();
   Thread thread;
   String smsPort;
   boolean firstTime;

   public PushMidlet() {
      display = Display.getDisplay(this);
      smsPort = getAppProperty("SMS-Port");
      firstTime = true;
      form = new Form("Event Listener");
      form.addCommand(cmdExit);
      form.addCommand(cmdAlarm);
      form.addCommand(cmdReg);
      form.addCommand(cmdUnreg);
      form.setCommandListener(this);
   }

   public void startApp() {
      getListConnections();
      display.setCurrent(form);
   }

   public void pauseApp() {
   }

   public void destroyApp(boolean unconditional) {
   }

   /*Check for list of connections recieved*/
   public void getListConnections() {
      String regConnections[];
```

```
    regConnections = PushRegistry.listConnections(true);

    if (regConnections.length != 0) {
      form.append("Launched using SMS Event to UN Register select UNRegister SMS");

      for (int i = 0; i < regConnections.length; i++) {
        try {
          MessageConnection msgconn = (MessageConnection)
Connector.open(regConnections[i]);
          msgconn.setMessageListener(this);
          allConn.addElement(msgconn);
        } catch (SecurityException exSec) {
          System.out.println("SecurityException=" + exSec);
        } catch (IOException exIO) {
          System.out.println("IOException==" + exIO);
        }
      }

    } else {
      if (firstTime) {
        form.append("Manually launched");
        form.append("Here you can select Register and Un Register SMS also Register Alarm
Dynamically");
        firstTime = false;
      }
      regConnections = PushRegistry.listConnections(false);
    }


  }

  /*Register the Alarm for certain period for auto launch*/
  private void registerAlarm(final long duration) {
    new Thread() {

      public void run() {

        long alarmTiming = System.currentTimeMillis() + duration;

        try {
          /*to register MIDlet for a time period*/
          PushRegistry.registerAlarm(midletName, alarmTiming);
        } catch (ClassNotFoundException ex) {
        } catch (ConnectionNotFoundException ex) {
        }
      }
    }.start();
```

```
}

public void commandAction(Command cmd, Displayable disp) {
   if (cmd == cmdExit) {
      exitMidlet();

   } else if (cmd == cmdAlarm) {
      registerAlarm(40000);
   } else if (cmd == cmdReg) {

      RegisterSMSConn();
   } else if (cmd == cmdUnreg) {
      UnRegisterSMSConn();
   }
}

public void exitMidlet() {
   closeConnections();
   destroyApp(true);
   notifyDestroyed();
}

/*Make dynamic connection registered for a specific sms port*/
public void RegisterSMSConn() {
   thread = new Thread(this);
   thread.start();

}

public void run() {
   try {
      /*To register Midlet for a port number*/
      PushRegistry.registerConnection("sms://:" + smsPort, midletName, "*");
      closeConnections();
      getListConnections();
   } catch (ClassNotFoundException exe) {
   } catch (IOException ex) {
   }
}

/*To unregister the port dynamically*/
public void UnRegisterSMSConn() {
   PushRegistry.unregisterConnection("sms://:" + smsPort);
}

public void closeConnections() {
```

```
    if (allConn != null) {
      while (allConn.isEmpty() == false) {
        MessageConnection msgConn =
            (MessageConnection) allConn.firstElement();
        if (msgConn != null) {
          try {
            msgConn.setMessageListener(null);
            msgConn.close();
          } catch (Exception exp) {
          }
        }
        allConn.removeElementAt(0);
      }
    }
  }

  /* Invokes when recieved message on specify port*/
  public void notifyIncomingMessage(MessageConnection msgConn) {

    TextMessage message = null;
    try {
      message = (TextMessage) msgConn.receive();
      String sendAddr = message.getAddress();
      form.setTitle(sendAddr);
      String messageText = message.getPayloadText();
      form.append(messageText);
    } catch (IOException ex) {
      form.append("Exception here" + ex);
    }

  }
}
```