Implementing Push technology with J2ME and MIDP

Presented by developerWorks, your source for great tutorials

ibm.com/developerWorks

Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

Section 1. Before you start

About this tutorial

With the Mobile Information Device Profile (MIDP) version 1.0, the Application Manager System (AMS) was the only means to start a MIDlet. Although the AMS is still ultimately responsible for installing, managing, starting, and stopping MIDlets, MIDP 2.0 applications can now be started based on a request from a remote connection or scheduled timer.

This new feature brings with it a wealth of potential applications. For example, a remote application can push stock quote information on an hourly basis to a mobile device. Or how about this -- notifications from the corporate office can be sent to road warriors to alert them of schedule changes. Using the alarm feature, a MIDlet can now offer up reminders of scheduled events, such as a reminder of a meeting an hour before it begins.

This tutorial will walk through the basics of using Push technology with MIDP 2.0, including development of a MIDlet that will be activated based on an incoming SMS message (Simple Message Service).

About the author

John Muchow is a freelance technical writer and recruiter. He is the author of *Core J2ME Technology and MIDP*, a best-selling J2ME book that has been translated into Chinese, Korean and Italian. Visit Core J2ME (see *Resources*) for additional source code, articles, and developer resources. For additional information about writing projects or technical recruiting, you can contact John at *john@corej2me.com*.
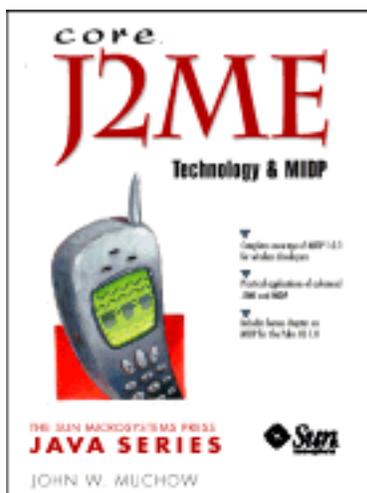
Section 2. Download and install the software

Required components

You'll need two different software tools before you can go any further:

- The Java Development Kit (JDK)
- The Wireless Toolkit (WTK)

Each of these tools is discussed in the next panels

Download the Java Development Kit

The JDK provides the Java source code compiler and a utility to create Java Archive (JAR) files. When working with version 2.0 of the Wireless Toolkit (as we are here), you will need to download JDK version 1.4 or greater.

Download JDK version 1.4.1.

Download the Wireless Toolkit

The Sun Microsystems Wireless Toolkit is an integrated development environment (IDE) for creating Java 2 Platform, Micro Edition (J2ME) applications, commonly referred to as *MIDlets.*

The WTK download contains an IDE, as well as the libraries required for creating MIDlets.

Download J2ME Wireless Toolkit 2.0.

This tutorial builds on an earlier *developerWorks* tutorial -- an excellent starting point if you are new to the Wireless Toolkit.

MIDlet development with the Wireless Toolkit.

Install the software

The Java Development Kit (JDK)
Use the JDK documentation to install the JDK. You can choose either the default directory or specify another directory. If you choose to specify a directory, make a note of where you install the JDK. During the installation process for the Wireless Toolkit, the software attempts to locate the Java Virtual Machine (JVM); if it cannot locate the JVM, you are prompted for the JDK installation path.

The Wireless Toolkit (WTK)

The Wireless Toolkit is contained within a single executable file, which you downloaded in the previous panel. Run this file to begin the installation process. It is recommended that you use the default installation directory. However, if you do not use the default directory, make sure the path you select does not include any spaces.

Section 3. MIDlet activation

Three ways to start a MIDlet

Once installed on a mobile device, a MIDlet can be activated in one of three ways:

- User request
- Incoming network connection
- Scheduled alarm

Let's quickly review network- and alarm-based activation.

Incoming network connection

The Application Manager System (AMS) can monitor port activity looking for a specific connection type and port. With the additional network connectivity options in MIDP 2.0, an incoming network request can be any of the following types:

- Message-based, such as SMS
- Packet-based, such as datagram
- Stream-based, such as socket

Scheduled alarm

In addition to an incoming network connection, a MIDlet can be activated at a specific time. This can be very useful for repeating events such as reminders for meetings or other appointments.

An alarm time is specified using a long integer, with the same format as the `Date.getTime()` method. The integer represents the number of milliseconds since January 1st, 1970.

Section 4. Registration

Types of registration

In order for a MIDlet to be notified of an incoming network connection or to "wake up" at a scheduled time, it must register these requests with the AMS. The AMS manages a list of acceptable incoming network connections and time-based alarms in what is referred to as the push registry. Requests are registered in one of two ways:

- Dynamically, at run time
- Statically, through entries in the Java Application Descriptor (JAD) file

Dynamic registration

Dynamic registration occurs when a MIDlet notifies the AMS at run time of its intent to allow incoming network connections or to be activated sometime in the future through an alarm.

Call the `registerConnection()` method in the PushRegistry class to register network connections,:

```
registerConnection(String connection, String midlet, String filter)
```

For alarm registration, prior to exiting the MIDlet, call the `registerAlarm()` method in the PushRegistry class:

```
registerAlarm(String midlet, long time)
```

Dynamic registration for incoming network connection

Below is a sample call to the `registerConnection(String connection, String midlet, String filter)` method.

```
PushRegistry.registerConnection("datagram://:2000", this.getClass().getName(), "*");
```

As with static registration, the connection URL is given the name of the MIDlet to invoke and any filter to be applied to the sender. Notice how I specify the current (active) MIDlet using the `getClass().getName()` method.

Dynamic registration for alarm

Below is a sample call to the `registerAlarm(String midlet, long time)` method.

```
Date now = new Date();
PushRegistry.registerAlarm(this.getClass().getName(), now.getTime + (1000 * 60 * 60));
```

This requests that the AMS activate this MIDlet one hour from the current time. Remember, the 'time' parameter is specified in milliseconds -- 3,600,000 milliseconds equals one hour.

Note: Only one alarm per MIDlet can be registered at any time.

Static registration

Incoming network requests can be defined in the JAD file. This type of request is considered static in that the sender and connection type are known when the MIDlet is installed. A static registration is defined using the `MIDlet-Push-<n>` attribute.

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

where:

- ConnectionURL is the connection string specifying the URL to monitor for incoming connection.
- MIDletClassName is the MIDlet class name to invoke.
- AllowedSender is the filter to specify what senders are allowed to request an incoming connection.

Examples of static registration

Examples of the `MIDlet-Push-<n>: <ConnectionURL>, <MIDletClassName>, <AllowedSender>` are:

```
MIDlet-Push-1: sms://:1000, PushDemo, *

MIDlet-Push-2: socket://:100, corej2me.NewsLink, *
```

When setting the `AllowedSender` parameter as an asterisk (*), the AMS accepts requests from any sender over the specified port. The specifics of each parameter will be shown as the tutorial progresses.

Note: Alarm-based activation is not supported through static registration.

Removing registration

Whenever a MIDlet suite is uninstalled, the AMS removes any static registrations. Should you need to remove a dynamic registration, call the `PushRegistry.unregisterConnection()` method specifying the URL to remove:

Example:

```
PushRegistry.unregisterConnection("datagram://:2000")
```

Note: Static registration entries can only be changed by uninstalling a MIDlet, making the changes to the JAD file, and re-installing the MIDlet. Because dynamic registration is performed at run time, changes can be made at any time.

Section 5. Example: Incoming network connection, Part 1

Overview

The following example walks you through creating, packaging, downloading, and testing a MIDlet that is activated by an incoming SMS message.

This MIDlet relies on static push registration. If you recall, this means an entry specifying the incoming network parameter must be added in the JAD file. Also, you will step through using WTK to simulate sending an SMS message to an emulated device.

The following example shows how an incoming message will look on the emulator when the application is complete.

Figure 1. SMS message

Creating the project

To create the project:

1. Start the wireless toolkit.
2. Select **New Project**.
3. Enter the values shown in Figure 2.
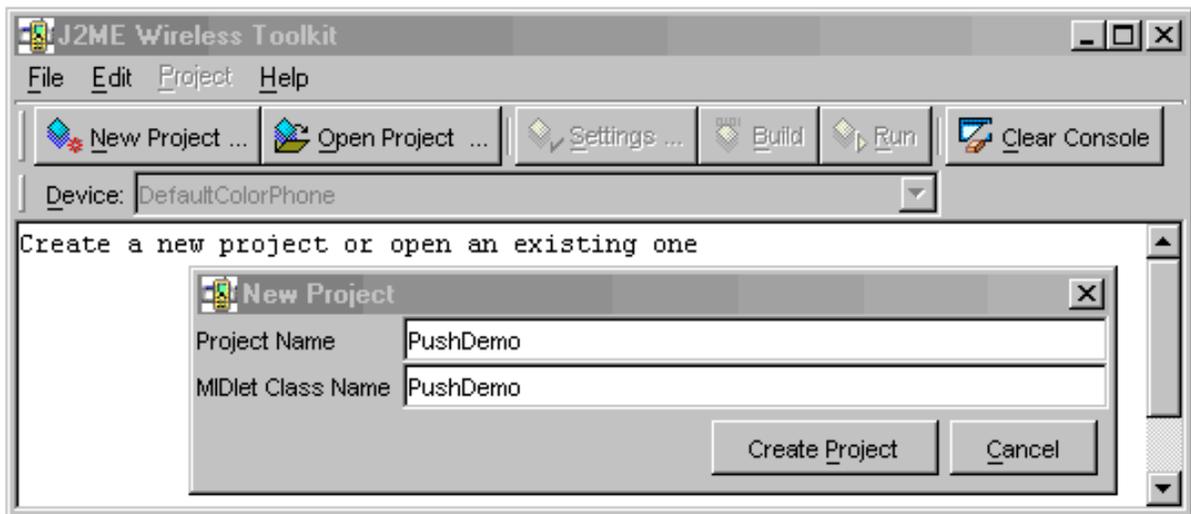
4. Click **Create Project**.
5. Click **OK** when the settings dialog box is displayed.

Writing the code

Simulating Over-the-Air (OTA)

As mentioned previously, J2ME-enabled devices are equipped with an Application Manager System (AMS) to assist in downloading, installing, and configuring J2ME applications. The WTK can simulate this process, which I will step through next.

Packaging the MIDlet

Before going any further, you need to package the MIDlet into a JAR file and create the associated JAD file. The steps to do this are:

1. Create the push registry entry.
2. Build the MIDlet.
3. Create the JAR and JAD files.

Create the push registry entry

This MIDlet uses static registration. Here is how to specify the attributes inside the JAD file:

1. Select **Settings** on the WTK application.

2. Select the Push Registry tab.
3. Click **Add**.
4. Enter the information shown in Figure 3 and select **OK**.

Figure 3. Push Registry
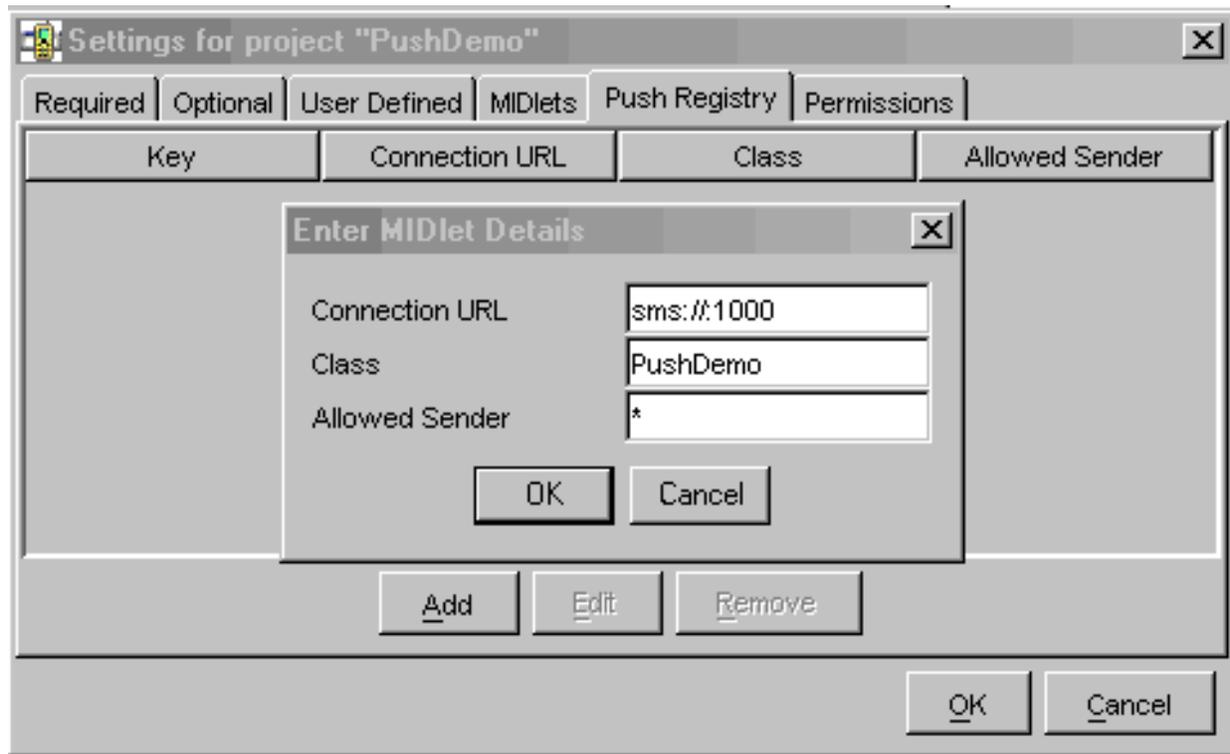
Where:

- The Connection URL refers to the port that is monitored for an incoming network connection.
- The Class refers to the MIDlet that will be invoked when a message arrives.
- With the Allowed Sender set to '*', the AMS does not perform any filtering on the incoming request (that is, any sender is allowed).

Build the MIDlet

To compile and preverify the MIDlet, select **Build** from the WTK toolbar.

Figure 4. Build MIDlet
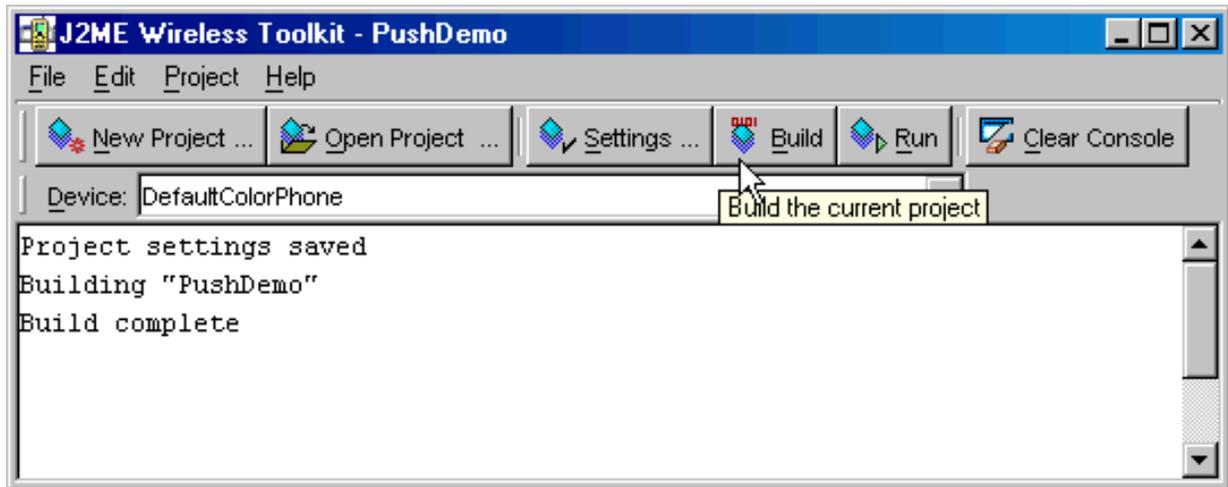
Create the JAR and JAD files

To package the MIDlet, select **Project>Package>Create Package** from the WTK toolbar.

Figure 5. Package
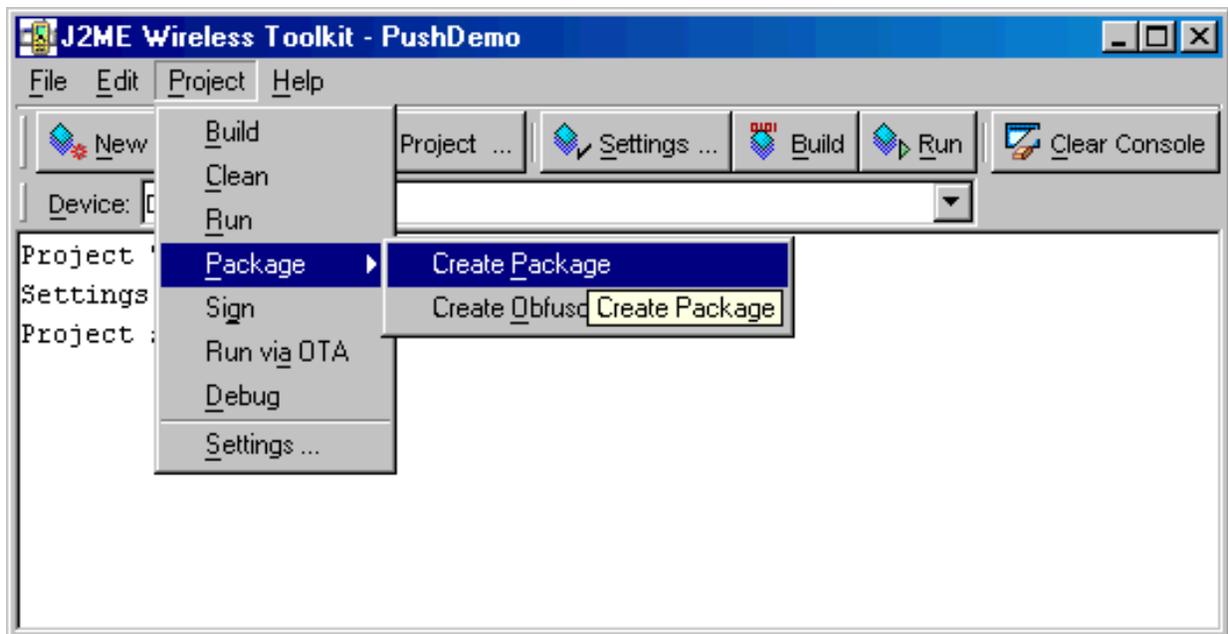
WTK pulls together all the information necessary to create both the JAR and JAD files. If you have ever created a JAR or JAD file the old-fashioned way, editing the files yourself, you'll truly appreciate the ease and accuracy with which WTK creates these files. You no longer need to concern yourself that the JAR file size specified in the JAD file entry MIDlet-Jar-Size matches the actual file size as created by the pre-verifier, as this is managed by WTK.

Reviewing the JAD file

Take a moment to view the settings within the JAD file. Work your way down the file hierarchy to the x:\WTK20\apps\PushDemo\bin directory. Here you'll find both the JAR and JAD files.

Figure 6. Review JAD file
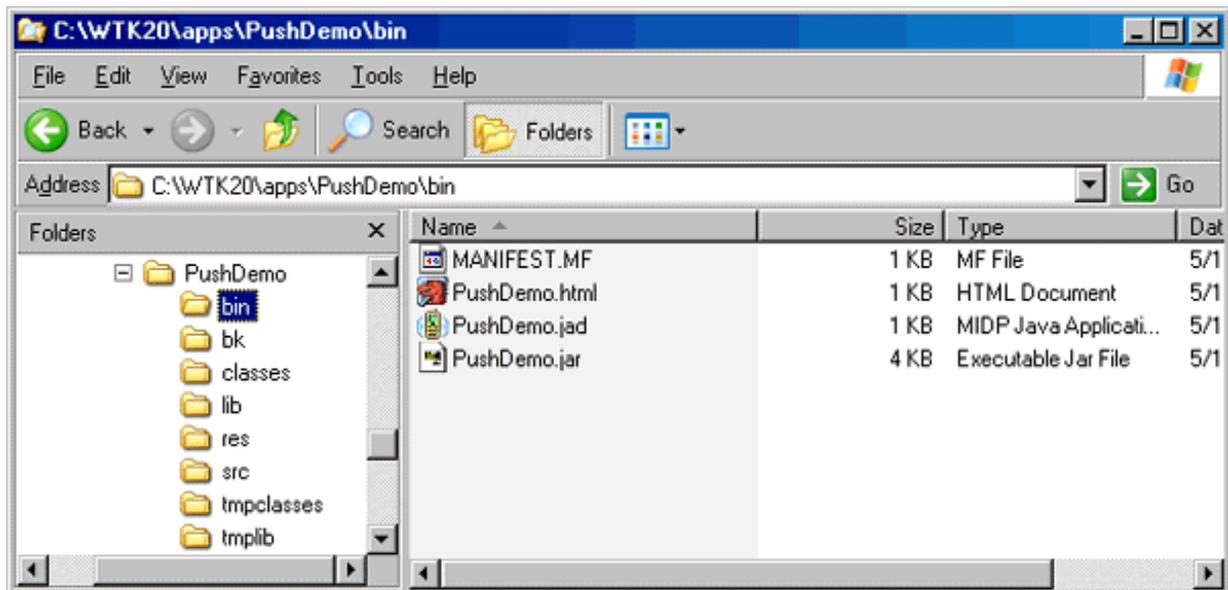
Looking inside the JAD file, you can see the push registry entry previously created.

```
MIDlet-1: PushDemo, , PushDemo
MIDlet-2: PushDemoAlarm, , PushDemoAlarm
MIDlet-Jar-Size: 3528
MIDlet-Jar-URL: PushDemo.jar
MIDlet-Name: PushDemo
MIDlet-Push-1: sms://:1000, PushDemo, *
MIDlet-Vendor: Sun Microsystems
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-2.0
```

At this point you have all that you need to move to the next step: testing an incoming connection.

Installing the MIDlet via OTA, Step 1

At this point in the "real world," the packaged files (JAR and JAD) would be placed on a remote machine. A J2ME-enabled device would use a built-in browser to access the URL where the files are located, and could select, download, and install the MIDlet. With WTK, you can

simulate this process.

Begin by selecting **Run via OTA** from the WTK Project menu. After the AMS starts, click **Apps**. Click **Menu** and select **Launch** to start the install application.

Figure 7. AMS Step 1
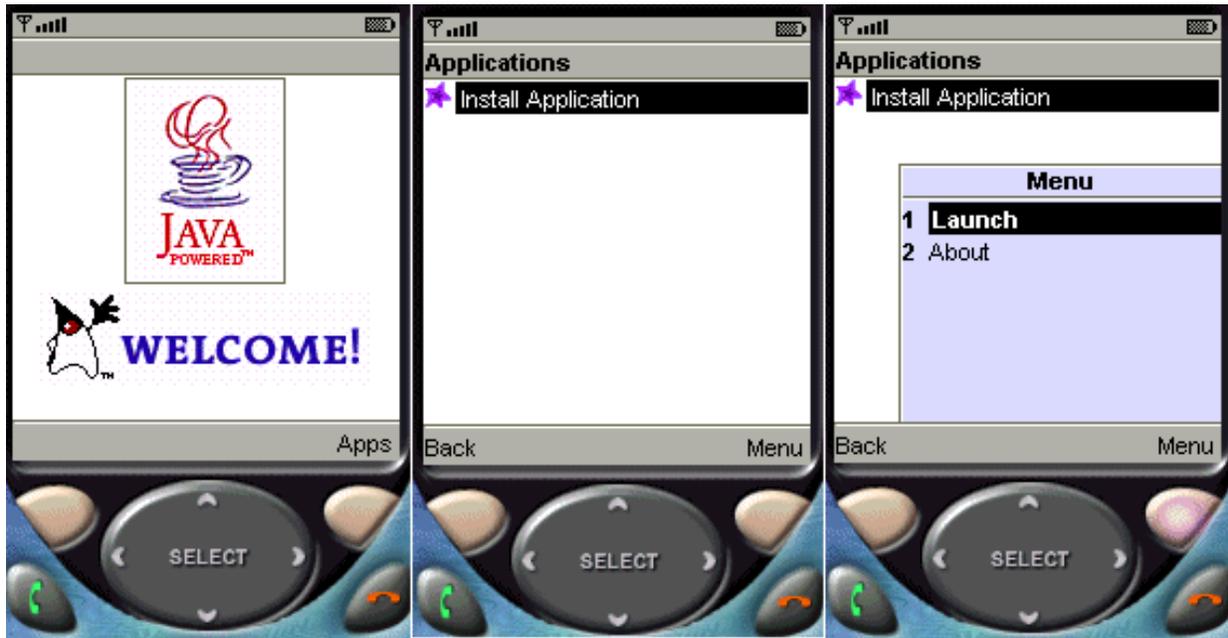
Installing the MIDlet via OTA, Step 2

The install application will present a URL, simulating access to a server that is hosting J2ME MIDlets. Select **Menu**, and select **Go**. This initiates a connection to the URL (simulating this action) and downloads a list of available MIDlets.

Figure 8. AMS Step 2

Installing the MIDlet via OTA, Step 3

The only MIDlet available will be the push example. Select **Install** to begin. The AMS always begins by downloading only the JAD file. This gives the AMS the opportunity to check software version information, available memory, and other configuration details. If for some reason the AMS cannot load the MIDlet, this is where you would be notified, and the installation process would end.

Click **Install** to begin downloading the JAR file.

Figure 9. AMS Step 3
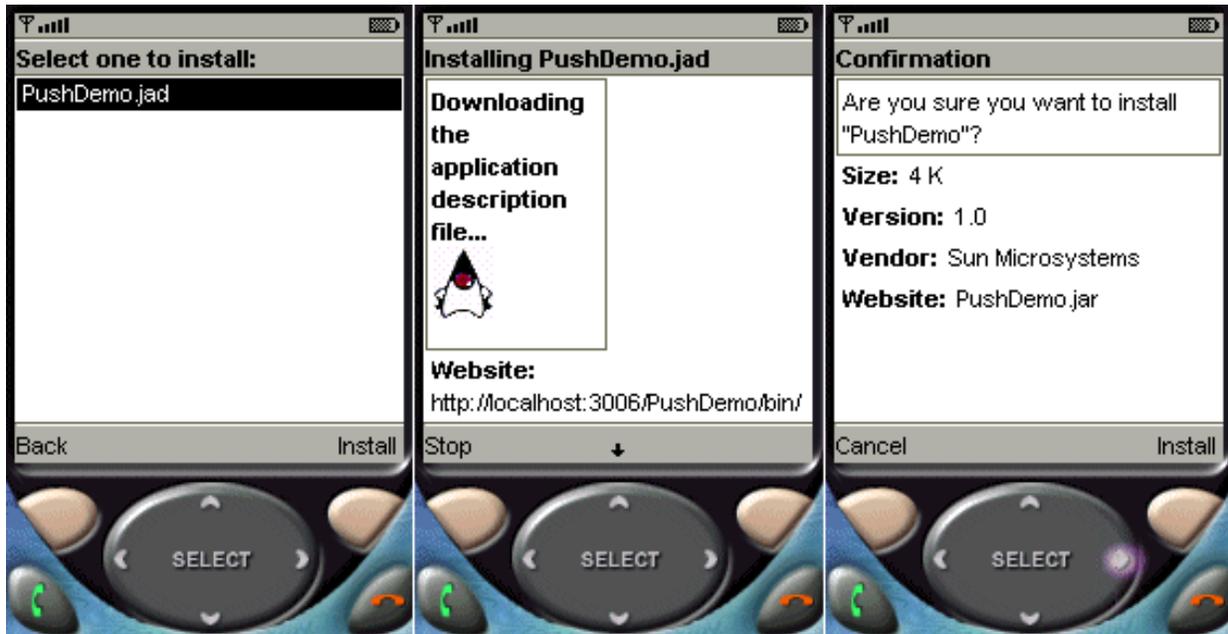
Installing the MIDlet via OTA, Step 4

Once the download begins, even though you are simulating the OTA process, you are prompted with two dialog boxes requesting permission to start the MIDlet based on incoming requests (our push functionality) and whether you are willing to accept charges for airtime when receiving incoming network data. Select **Yes, always. Don't ask again** for both scenarios.

Figure 10. AMS Step 4
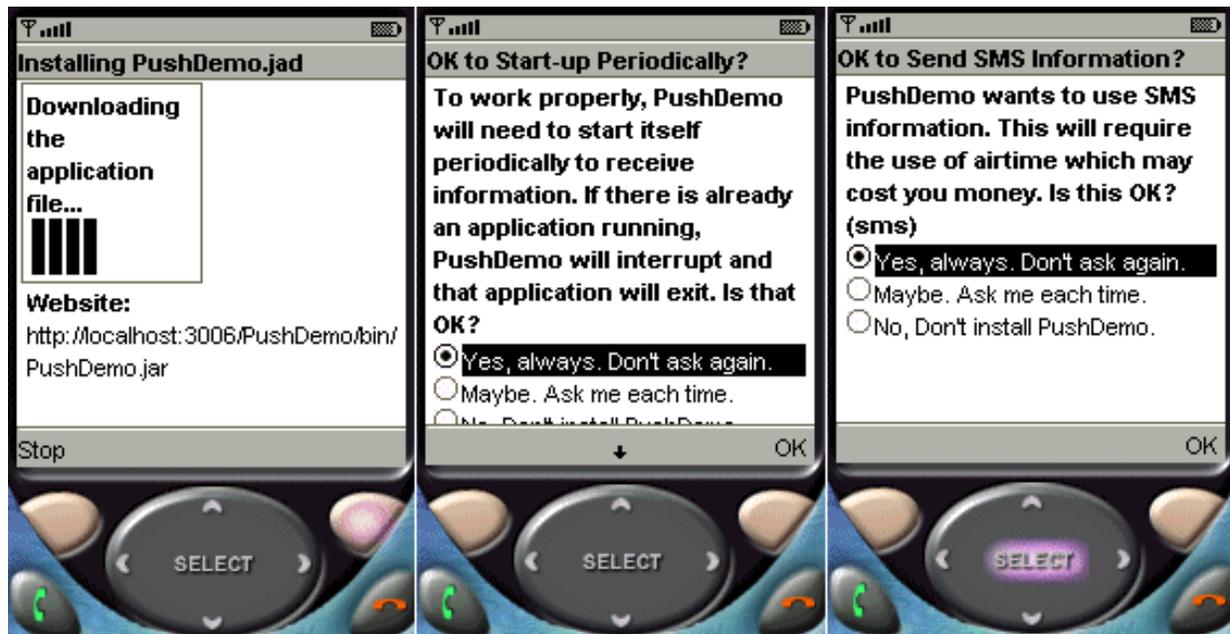
Installing the MIDlet via OTA, Step 5

With the download complete, you are returned to the AMS applications menu. Select **Back** to return to the AMS start screen.

At this point, the device (emulator) is in an idle state. If this were an actual mobile device, it would be analogous to 'main' display where no applications are active. This is important to test the functionality of the MIDlet. You want to know that the MIDlet is not running, and will be started based on an incoming network request.

Figure 11. AMS Step 5
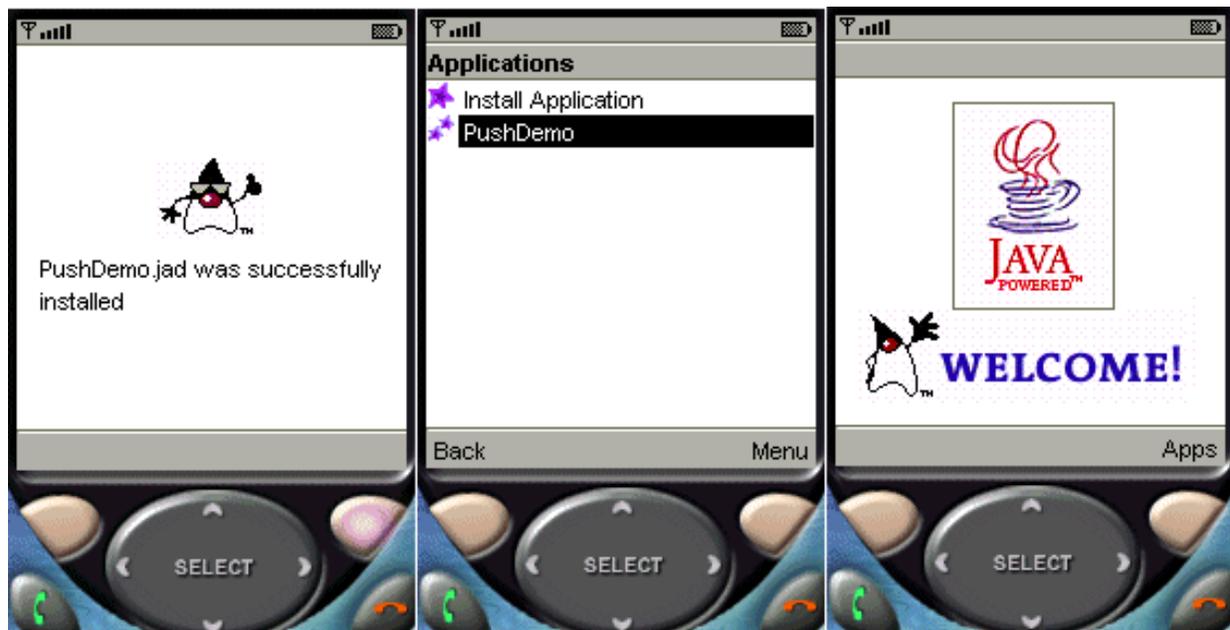
Installing the MIDlet via OTA, Step 6

One final note before moving on. You need to make note of the phone number of this device so you know where to send the SMS message. In Figure 12, notice the phone number in the upper left corner -- in this example the number is +5550001. Jot down the value displayed on your device.

Figure 12. AMS Step 6

Section 6. Example: Incoming network connection, Part 2

Open messaging console

With the MIDlet packaged, downloaded and installed, you are now ready to send an SMS message to activate the MIDlet. WTK provides the ability to simulate sending SMS messages. To open the console from which you can send a message, select **Utilities** from within the File menu.

Figure 13. File Utilities
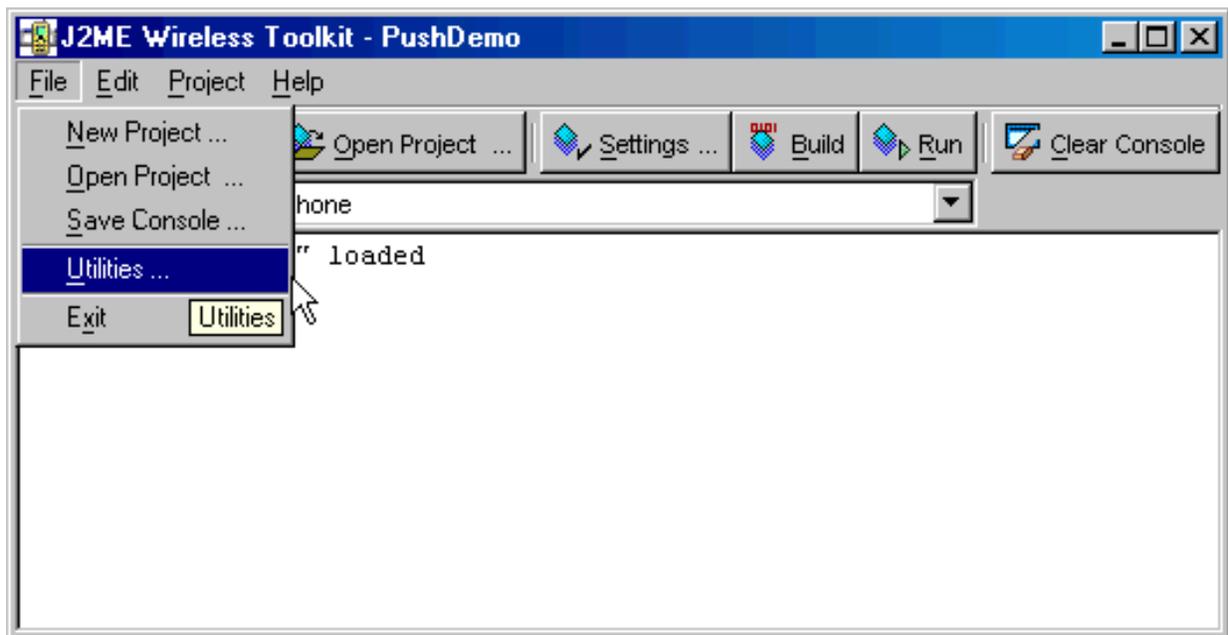
Once the Utilities dialog box appears, select **Open Console** under WMA (Wireless Messaging API).

Figure 14. WMA Console
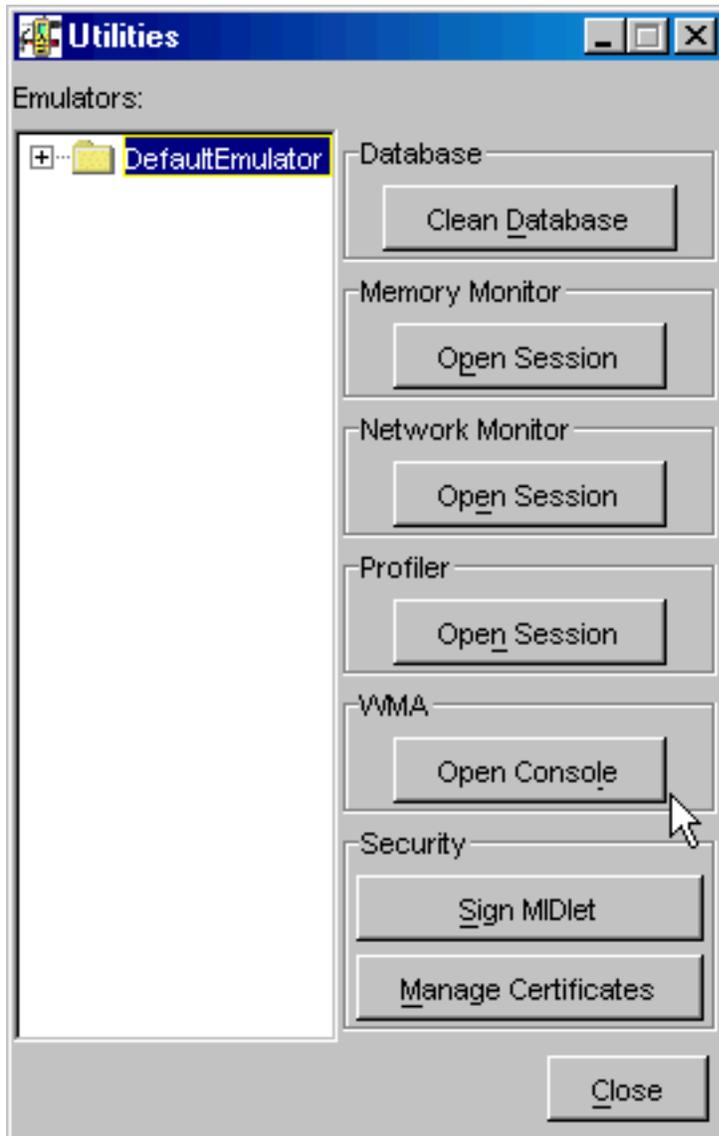
Entering the SMS message

Once the console appears, select **Send SMS** message.

Figure 15. WMA 1
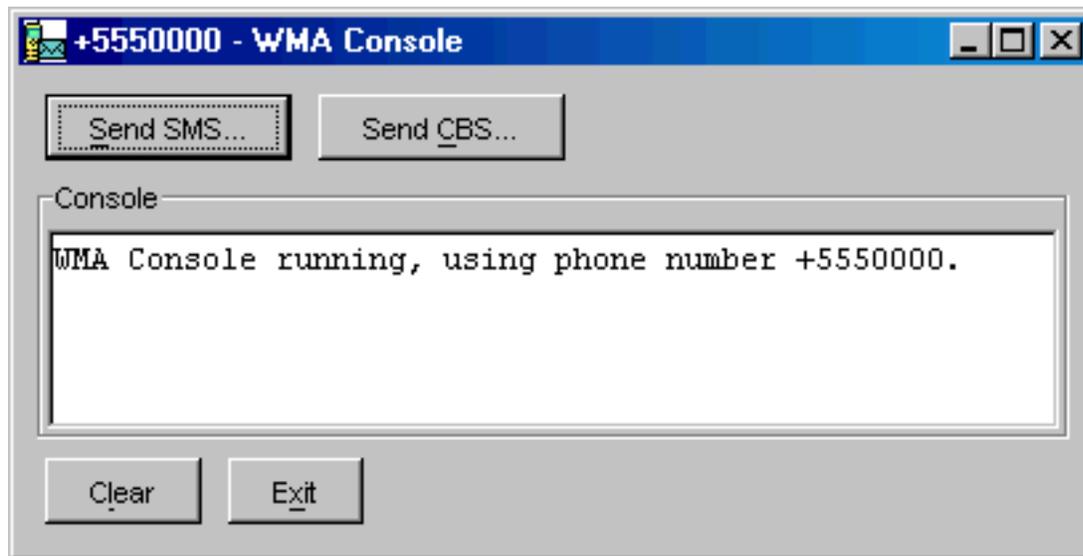
It is important to select the phone number of the client that matches the phone number of your emulator. If your device phone number does not appear in the list, click **Add Unlisted Client** to add the number. Also, the port number must be set to 1000, which is the value defined previously in the push registry. With everything now in place, type a message and press **Send**.

Figure 16. WMA 2
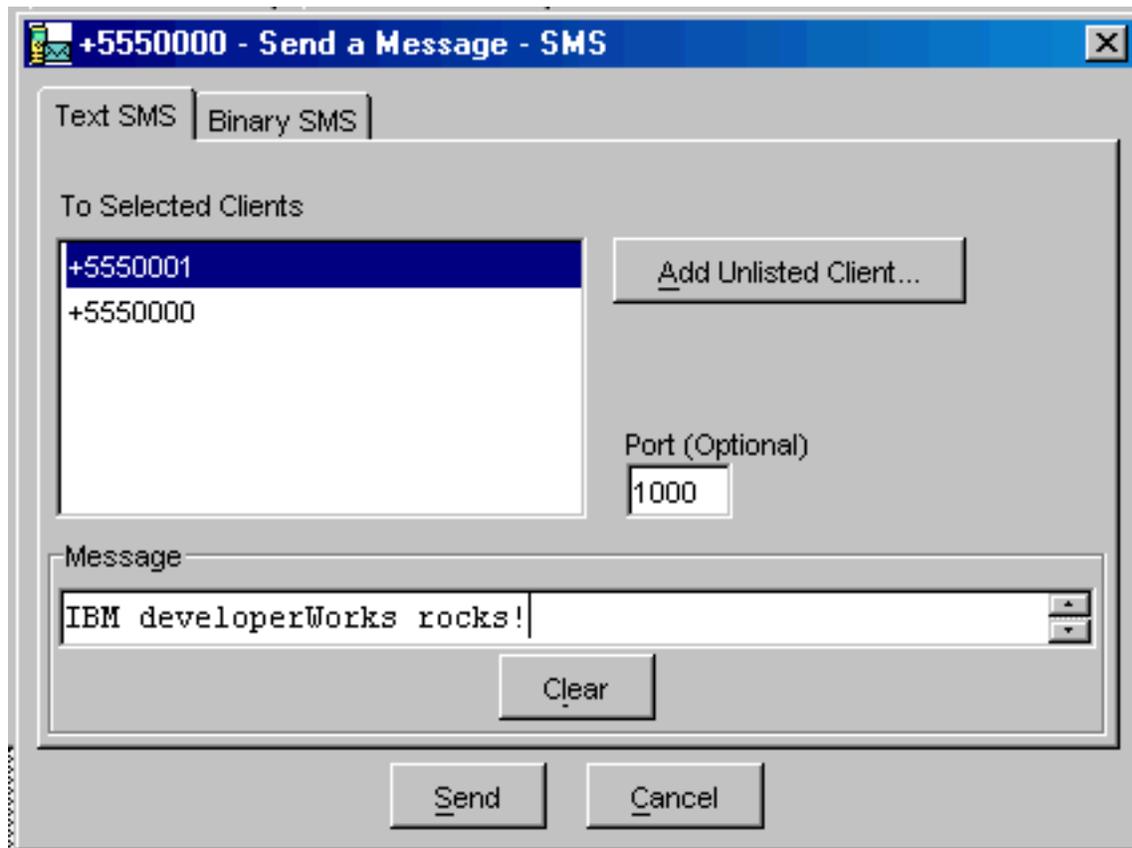
Approve incoming data request

Once the emulator recognizes there is an incoming connection request, it prompts you again for permission to accept incoming messages. Select **Yes, always, Don't ask again**. With this setting, any additional SMS messages sent to the device will not require answering this question again.

Figure 17. WMA 3
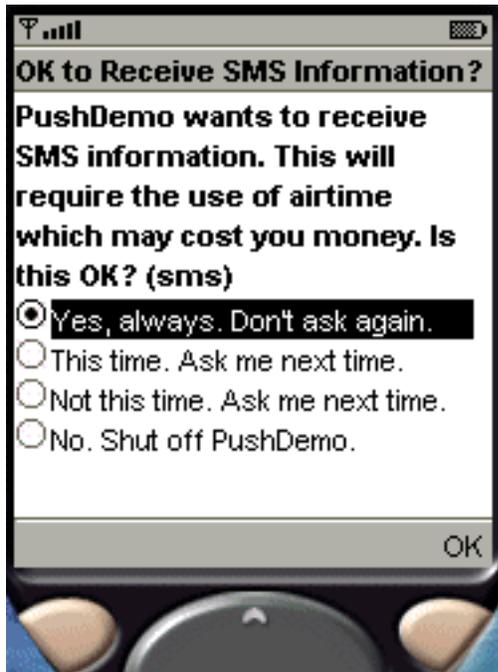
MIDlet activation and displaying the incoming message

Finally, with the SMS message in-hand, AMS activates the MIDlet through a call to `startApp()` and displays the message.

Figure 18. WMA 4

Section 7. Reviewing the code

Class definition

Take a few moments to review the code. Below is the class definition of the MIDlet.

```
public class PushDemo extends MIDlet implements CommandListener
{
  private Display display;                 // Reference to display
  private Command cmExit;                  // Command to exit
  private Alert alertIncomingMessage;      // Incoming message
  int incomingPortNum = 1000;              // Port to listen for connection
  MessageConnection incomingConnection = null;  // Connection for receiving the message
  Message incomingMessage;                 // Incoming message

    ...

}
```

A few points worth noting:

- The Alert is used to display the incoming SMS message.
- The incomingPortNum can be any port that is not currently in use.
- The incomingConnection is the network connection.
- The incomingMessage refers to the incoming data.

Constructor

There are two primary steps when allocating an instance of this MIDlet class: get a reference to the display and allocate a modal Alert that displays the incoming SMS as well as listens for an event to exit the MIDlet.

```
public PushDemo()
{
  display = Display.getDisplay(this);

  alertIncomingMessage = new Alert("Incoming Message");
  alertIncomingMessage.setTimeout(Alert.FOREVER);

  cmExit = new Command("Exit", Command.EXIT, 1);
  alertIncomingMessage.addCommand(cmExit);

  alertIncomingMessage.setCommandListener(this);
}
```

Check for incoming connection

Let's see what transpired inside the MIDlet to receive and display the incoming message.
Looking inside the `startApp()` method, the first check is to see if the MIDlet was invoked
based on an incoming connection. If it wasn't, simply exit the MIDlet:

```
public void startApp()
{

  String connectList[];

  // Was the MIDlet started by an incoming connection?
  connectList = PushRegistry.listConnections(true);
  if (connectList == null || connectList.length == 0)
  {
    // Started by the user, exit
    destroyApp(false);
    notifyDestroyed();
  }
  else
    ...
```

The `true` parameter to the `listConnections()` method specifies to return only the
connections that have input (data) available.

If from a remote resource

If the connection was from a remote sender, the code in the else block is run:

```
 if (connectList == null || connectList.length == 0)
 {
   // Started by the user, exit
   ...
 }
 else  // Stared from an incoming connection...
 {
   try
   {
     incomingConnection = (MessageConnection) Connector.open("sms://:" + incomingPortNum);


     // Receive the message
     incomingMessage = incomingConnection.receive();

     // If it's a text message, add it to the alert
     if(incomingMessage != null && incomingMessage instanceof TextMessage)
     {
       alertIncomingMessage.setTitle("From: " + incomingMessage.getAddress());
       alertIncomingMessage.setString(((TextMessage)incomingMessage).getPayloadText());

         // Display the message
```

```
        display.setCurrent(alertIncomingMessage);
    }
  }
  catch(IOException e)
  {
    System.out.println("IO Exception!");
  }
```

The `Connector.open()` call references the `ConnectionURL` parameter previously placed in the JAD file:

```
MIDlet-Push-1: sms://:1000, PushDemo, *
```

Once the message has been received, the title and message of an Alert component are set to the address of the sender and the incoming text message, respectively. The final step is to set the current displayable to the Alert just created, which will display the incoming SMS message on the device!

Section 8. Summary and resources

Summary

WTK offers a range of tools for developing push applications. In this tutorial I discussed how MIDlets created with MIDP 2.0 can be activated through an incoming network connection or a scheduled alarm. I also presented the two registration options, dynamic and static, which notify the AMS of the intent to activate a MIDlet.

Most of this tutorial dealt with creating an application that is activated based on receiving an SMS message. There are many steps to package, deploy, install, and test a push-based MIDlet. However, with WTK the process is greatly simplified with the abundance of built-in tools that support working with and simulating almost every process along the way.

Resources

- Visit *Core J2ME* for additional source code, articles, and developer resources.
- Refer to the *developerWorks* tutorial *MIDlet development with the Wireless Toolkit* to see how this tutorial built on the previous tutorial.
- For information on the Java Development Kit 1.4.1 refer to the *JDK*.
- The *J2ME Wireless Toolkit* can be located here.
- For information on the Java Specification Request for Wireless Messaging API see *JSR 120*.
- Information on the *Java Community Process* can be found here.

Feedback

Please let us know whether this tutorial was helpful to you and how we could make it better. We'd also like to hear about other tutorial topics you'd like to see covered.

For questions about the content of this tutorial, contact the author, John Muchow, at *John@CoreJ2ME.com*.

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The open source Toot-O-Matic tool is an XSLT stylesheet and several XSLT extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML. (It also saves our production team a great deal of time and effort.)

You can get the source code for the Toot-O-Matic at
www6.software.ibm.com/dl/devworks/dw-tootomatic-p. The tutorial Building tutorials with the
Toot-O-Matic demonstrates how to use the Toot-O-Matic to create your own tutorials.
developerWorks also hosts a forum devoted to the Toot-O-Matic; it's available at
www-105.ibm.com/developerworks/xml_df.nsf/AllViewTemplate?OpenForm&RestrictToCategory=11.
We'd love to know what you think about the tool.