MIDlet development 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. Introduction

Why J2ME?

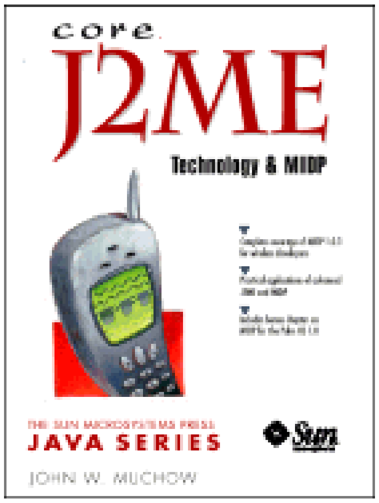
Nokia, the dominant player in the mobile phone market, has estimated that in 2002 it will sell over 50 million J2ME-enabled mobile phones. With lofty numbers announced by other mobile phone manufacturers as well, there is a flood of interest in software development revolving around J2ME (Java 2 Micro Edition). J2ME is a slimmed-down version of Java targeted at devices that have limited memory, display, and processing power.

This tutorial provides a step-by-step introduction to downloading, installing, and configuring your computer to begin development of applications for this runtime environment.

There will be a specific focus on application development for mobile devices using an application programming interface (API) known as the Mobile Information Device Profile (MIDP). Applications written for this API are affectionately referred to as MIDlets.

About the author

John Muchow is the author of "*Core J2ME Technology and MIDP*", currently the best selling J2ME book at Amazon.com. Published by Sun Microsystems Press, it is a recent addition to the popular Java Series.



John writes extensively about J2ME and maintains the wireless developer Web site

Core J2ME. The site contains an abundance of source code, articles, developer tools, and resources, and links to the latest releases of J2ME software.

The Wireless Mind, Inc.

John is also founder of *The Wireless Mind, Inc.,* a business that provides training for wireless developers, specializing in J2ME. In the first quarter of 2002, The Wireless Mind will introduce online (Web-based) J2ME training.

Section 2. The basics

Configurations

To support a broad range of devices, from the smallest of mobile phones to much more capable devices, such as a television set-top box, J2ME introduced two concepts: the *configuration* and *profile*.

Configurations consist of the Connected Device Configuration (CDC) and the Connected Limited Device Configuration (CLDC).

The CDC runs the "classic" Java virtual machine (VM). That is, it consists of the same functionality available for a VM running on a desktop computer. Several megabytes of memory are necessary to support this VM. Devices such a set-top box, or an Internet screen phone (for online banking, e-mail, etc.) would fall into this configuration.

The CLDC is intended for devices with limited hardware capabilities, such as mobile phones, low-end personal digital assistants (PDAs), and pagers.

CDC and CLDC

Following are general characteristics for each configuration:

Connected device configuration

- 512 kilobytes (minimum) memory for Java runtime
- 256 kilobytes (minimum) for runtime memory allocation
- Network connectivity, possibly persistent (always on), high bandwidth

Connected Limited Device Configuration

- 128 kilobytes memory for Java runtime
- 32 kilobytes memory for runtime memory allocation
- Limited user interface
- Limited power -- typically battery
- Network connectivity -- typically wireless

Profiles

A *profile* is an extension, of sorts, to a configuration. It provides a library of code to support a particular type, or range, of devices. Shown here are the specifics for two profiles, each defined through the Java Community Process (see Resources and additional information on page 24).

Mobile Information Device Profile -- Mobile and voice communication devices:

- 512K total memory for Java runtime and libraries
- Limited power -- typically battery
- Connectivity to wireless network
- Limited user interface

PDA profile -- small, resource-limited handheld devices:

- No less than 512KB total memory for Java runtime and libraries
- No more than 16 megabytes of memory
- Limited power -- typically battery
- Limited user interface with a minimum of 20,000 pixels
- User interface supports pointing device and character input

CLDC and MIDP have been widely accepted by many device manufacturers. Installation, configuration, and application development with CLDC/MIDP are the focus of this tutorial.

Section 3. Downloading the software

Required components

Three software tools are needed to develop MIDlets:

- Java Development Kit (JDK)
- Connected Limited Device Configuration
- Mobile Information Device Profile

Java Development Kit

The JDK provides the Java source code compiler and a utility to create Java Archive (JAR) files.

You can download the JDK version 1.3 at *http://java.sun.com/products/jdk/1.3*.

Connected Limited Device Configuration

CLDC provides a subset of the Java 2 Standard Edition (J2SE) libraries. Included are classes from:

- java.io
- java.lang
- java.util

An additional set of classes is provided by the CLDC -- **javax.microedition.io**. This library provides classes and interfaces to facilitate access to storage and network systems. Although this library lays the groundwork for accessing various systems, the implementation is provided at the Profile level. For example, MIDP is required to provide support for the HTTP protocol. It does so by extending a class within **javax.microedition.io**.

You can download CLDC version 1.0.3 from *http://java.sun.com/products/cldc*.

Mobile Information Device Profile

As mentioned previously, the libraries specific to a range of devices are implemented at the profile level. The MID Profile was created to support devices with limited screen, memory, and processing power.

You can download MIDP version 1.0.3 from *http://java.sun.com/products/midp*.

Section 4. Installing and configuring

Java Development Kit

Install the JDK as directed in the download. If you install the software in a directory other than the default, make note of the location. We will use this directory location when configuring the environment of your computer.

Connected Limited Device Configuration

I recommend that you extract the files in the CLDC download into a directory with the name **c:\j2me**. If you don't have space on your **C:** drive, substitute an appropriate drive letter.

If you downloaded version 1.0.3, files will be written into a subdirectory with the name: **j2me_cldc**.

The final, complete path to the CLDC installation is:

c:\j2me\j2me_cldc

Mobile Information Device Profile

As with the CLDC, I recommend you extract the files in the MIDP download into a directory with the name **c:\j2me**. Keeping CLDC and MIDP files together (within the same subdirectory) makes for better organization and will ease the pain should an upgrade be available in the future.

If you downloaded version 1.0.3, files will be written into a subdirectory with the name: **midp1.0.3fcs**.

The final, complete path to the MIDP installation is:

c:\j2me\midp1.0.3fcs

Updating the system environment

With the software installed, we need to add/update a few variables in the system environment. The **PATH** variable will need to reference several executable files in both JDK and MIDP. We'll also add/update the **CLASSPATH** variable to reference the class files of MIDP. Finally, we'll create a new environment variable, **MIDP_HOME**, that will point to several MIDP configuration files.

PATH variable

Update the PATH variable (create the variable if it does not exist) to include references to the **\bin** directories of the JDK and MIDP. For example:

PATH=c:\jdk1.3.1\bin;c:\j2me\midp1.0.3fcs\bin

This setting is based on the following:

- JDK version 1.3.1 installed into the directory **c:\jdk1.3.1**
- MIDP files extracted into the directory **c:\j2me**.

CLASSPATH variable

The CLASSPATH references where the Java compiler should search for classes that are not part of the JDK installation. We need a reference to the MIDP classes. Update the CLASSPATH variable (create the variable if it does not exist) to reference the **\classes** directory of MIDP. For example:

CLASSPATH=c:\j2me\midp1.0.3fcs\classes;.

There is a "**.**" at the end of the path to represent the current working directory. In other words, the Java compiler can look for classes that are included with MIDP, as well as in the current directory (where the compiler was invoked).

MIDP_HOME variable

This variable will reference the **\lib** directory in the MIDP installation. Create the variable as follows:

MIDP_HOME=c:\j2me\midp1.0.3fcs

Notice that we **do not** append **\lib** to the path. The MIDP software will handle this internally when accessing the files.

There are two configuration files located here:

- internal.config
- system.config

Take a peek inside these files to get an idea of the variables that may be configured. As an example, the variable **system.display.screen_depth** in the file internal.config can be set to reflect how many colors (or shades of gray) the mobile phone emulator will support.

MIDP defined screen values:

- 1 => Black and white
- 2 => 4 shades of gray
- 4 => 16 shades of gray
- 8 => 256 colors

Section 5. Your first MIDlet

Writing the code

Following is the code for a simple MIDlet.

This MIDlet will display a TextBox, which is a multiline, editable field. There will also be a button on the display to exit the MIDlet. Don't get too caught up with code that looks unfamiliar. What's important at this point is to understand the development cycle.

Using a text editor, enter the code, and save in a file with the name ***FirstMIDlet.java***.

```
/*--------------------------------------------------
* FirstMIDlet.java
*-------------------------------------------------*/
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class FirstMIDlet extends MIDlet implements CommandListener
{
  private Display display = null; // Reference to Display object
  private TextBox tbMain;         // A Textbox to display a message
  private Command cmExit;         // A Command to exit the MIDlet

  public FirstMIDlet()
  {
    // Command to the exit the MIDlet
    cmExit = new Command("Exit", Command.SCREEN, 1);

    // Create a textbox component
    tbMain = new TextBox("TextBox", "Welcome to J2ME and MIDP!", 50, 0);

    // Add the command onto the textbox
    tbMain.addCommand(cmExit);

    // Listen for events
    tbMain.setCommandListener(this);
  }

  // Called by application manager to start the MIDlet.
  public void startApp()
  {
    if (display == null)
      display = Display.getDisplay(this);
    display.setCurrent(tbMain);
  }

  // A required method
  public void pauseApp()
  { }
  // A required method
  public void destroyApp(boolean unconditional)
  { }

  // Check to see if our Exit command was selected
  public void commandAction(Command c, Displayable s)
  {
    if (c == cmExit)
    {
      destroyApp(false);
      notifyDestroyed();
    }
  }
}
```

Compiling the code

From a command prompt, in the directory where you saved the file FirstMIDlet.java, enter the following to compile the source code:

```
javac -bootclasspath c:\j2me\midp1.0.3fcs\classes
FirstMIDlet.java
```

The *-bootclasspath* parameter notifies the compiler where to locate the MIDP startup classes.

The resulting class file will be written into the current directory (the same location as FirstMIDlet.java).

Pre-verifying

Before running a MIDlet, the class file must be pre-verified. This is an extra step beyond what is required when writing a "traditional" Java application. Pre-verification is necessary to provide a quicker and more simplified class file verification once the files are loaded onto a mobile device. Class files that have not been pre-verified will fail to run.

From a command prompt, in the directory where you saved the file FirstMIDlet.java, enter the following to pre-verify the class file:

```
preverify -classpath c:\j2me\midp1.0.3fcs\classes;. -d .
FirstMIDlet
```

The *-classpath* parameter specifies where the pre-verifier is to look for classes. Specifically, the MIDP classes are located at `c:\j2me\midp1.0.3fcs\classes` , and the source code class file is in the current directory, specified with "**.**"

The *-d* parameter tells the pre-verifier to write the resulting class file into the current directory.

A note about pre-verifying

With the command line parameters, as previously shown, for compiling and pre-verifying, the pre-verified class file will overwrite the class file created by the Java compiler. This is perfectly fine. However, as your projects grow in complexity, it may be a good idea to write class files created by the compiler into one directory, and the pre-verified class files into another.

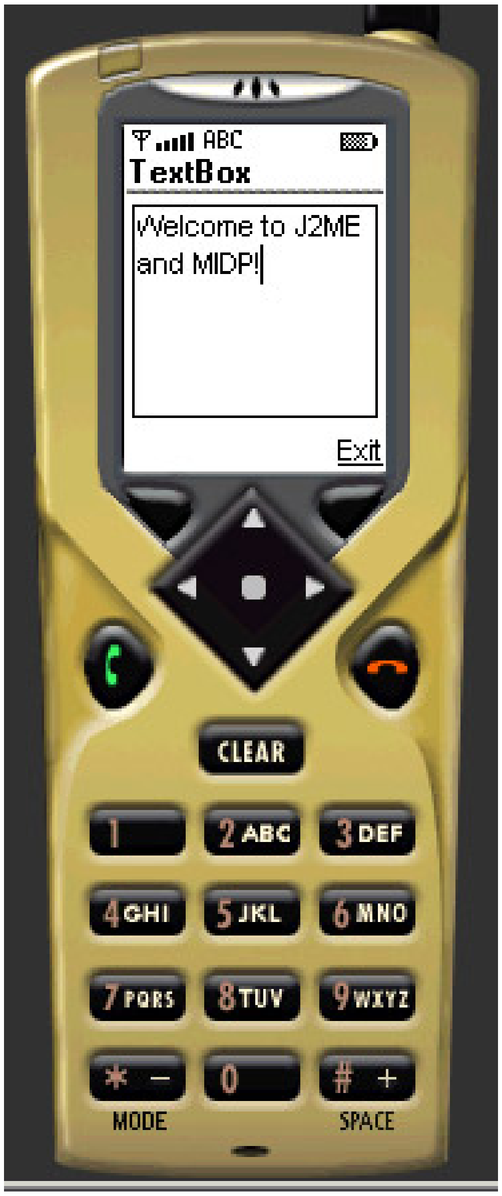
In the section Parting shots on page 23 , we'll see how the command line parameters have to change for both the compiler and pre-verifier to keep class files separate.

Running the MIDlet

Enter the following at the command prompt to load your MIDlet onto a mobile device emulator:

```
midp -classpath . FirstMIDlet
```

This instructs the emulator to look in the current directory (".") for pre-verified classes. The specific class to locate is FirstMIDlet. The emulator should look similar to this:


ABC
TextBox
Welcome to J2ME and MIDP!
Exit
CLEAR
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO
7 PQRS
8 TUV
9 WXYZ
* -
0
# +
MODE
SPACE

Section 6. The MIDlet suite

The basics

If, at some point, you would like to package your MIDlet(s) and transfer the same onto a device that supports MIDP, you will have to create a Java archive file (JAR). A JAR file consists of class files and any additional resources required by your application. Typical resources include images and application data.

A MIDlet suite is a JAR file that contains one or more MIDlets.

A Java Application Descriptor (JAD) is an optional file that can accompany the JAR when deploying your MIDlet(s). A JAD file provides information about MIDlets within a JAR. Although not a requirement, a JAD file may come in handy if a device is being installed in a MIDlet suite. The JAD makes information available to the application manager, the software on a MIDP-enabled device. With this information, the manager can determine if the device can accomodate the MIDlet.

For instance, the JAD file contains an attribute that specifies the size of the JAR file in bytes, containing the MIDlet. Armed with this information, the manager could peek at the memory availability on the device to see if there is adequate space available before downloading the JAR.

Inside the JAR

A JAR file consists of class files and any additional resources required by a MIDlet. There is one additional file that is contained in every JAR, called the *manifest*. We'll specify the contents of the manifest when we create the JAR, and we'll see how this is accomplished in just a moment. First, there are six attributes that **must** be in the manifest:

- **MIDlet-Name** -- Name of the MIDlet suite
- **MIDlet-Version** -- Version number of the MIDlet
- **MIDlet-Vendor** -- Who created the MIDlet
- **MIDlet-<n>** -- Information about MIDlet(s) in the suite
- **MicroEdition-Profile** -- What profile is required by the MIDlet
- **MicroEdition-Configuration** -- What configuration is required by the MIDlet

---

Simple manifest file

A simple manifest file may look like this:

```
MIDlet-Name: Memopad
MIDlet-Version: 1.0
MIDlet-Vendor: Core J2ME
MIDlet-1: Memopad, /images/Memopad.png, MainMemopad
MicroEdition-Profile: MIDP-1.0
MicroEdition-Configuration: CLDC-1.0
```

A few key points:

- The MIDlet-1 reference has three possible parameters:

1. The name of the MIDlet (e.g. Memopad)
2. The image file to associate with the MIDlet (e.g. /images/Memopad.png)

   *The only supported image format in MIDP 1.0 is PNG (see Resources and additional information on page 24).
3. The name of the class file to load the MIDlet (e.g. MainMemopad)

- The image reference (e.g. /images/Memopad.png) is an optional parameter.
- There may be multiple MIDlets within a suite.

  Put the numeric value in increments to refer to additional MIDlets. For example:

  `MIDlet-2: TodoList, /images/TodoList.png, MainTodoList`
- All six attributes must exist in the manifest file.

Inside the JAD file

A JAD file provides information about the MIDlet(s) within the suite. As with the manifest file (that is part of the JAR file) there are several required attributes that must be included in the JAD file.

Required JAD attributes:
- **MIDlet-Name** -- Name of the MIDlet suite
- **MIDlet-Version** -- Version number of the MIDlet
- **MIDlet-Vendor** -- Who created the MIDlet
- **MIDlet-<n>** -- Information about the MIDlet(s) in the suite
- **MIDlet-Jar-URL** The URL of the JAR file

- **MIDlet-Jar-Size** The size, in bytes, of the JAR

Note:

Although the MIDP specification does not state the attribute **MIDlet-<n>** as required, you may have trouble running your MIDlets without this attribute in the JAD file. Specifying this attribute does not cause any additional overhead; therefore, to maintain consistency, I recommend you always place this attribute in both the manifest and the JAD.

Simple JAD file

A simple JAD file may look like the following:

```
MIDlet-Name: Memopad
MIDlet-Version: 1.0
MIDlet-Vendor: Core J2ME
MIDlet-1: Memopad, /images/Memopad.png, MainMemopad
MIDlet-Jar: http://www.CoreJ2ME.com/MIDlets/MemoPad.jar
MIDlet-Jar-Size: 14073
```

A few key points:

- Notice that **MIDlet-Name, MIDlet-Version** and **MIDlet-Vendor** have the same values as the attributes with the same name in the manifest. This is a requirement.
- If an attribute other than the three just mentioned is shown in both the manifest and the JAD, the value inside the JAD will take precedence. For example, if both the manifest and JAD have a reference to **MIDlet-Jar-Size**, the value in the JAD file will be used by the application manager.

Creating a second MIDlet

Before we create a MIDlet suite, let's write one more MIDlet. This will give us the opportunity to package two MIDlets within our suite. Once the suite is loaded onto a mobile device emulator, we'll see how the application manager handles multiple MIDlets.

The next MIDlet will display a List component comprised of two entries. As before, don't worry about code that looks unfamiliar. Concentrate instead on how we put all the pieces together to compile, pre-verify, and package the MIDlets.

Using a text editor, enter the code below and save it in the same directory as the first MIDlet; use the name ***SecondMIDlet.java***.

```
/*--------------------------------------------------
* SecondMIDlet.java
*-------------------------------------------------*/
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class SecondMIDlet extends MIDlet implements CommandListener
{
  private Display display = null; // Reference to Display object
  private List lsMain;            // A List of items
  private Command cmExit;         // A Command to exit the MIDlet

  public SecondMIDlet()
  {
    // Command to the exit the MIDlet
    cmExit = new Command("Exit", Command.SCREEN, 1);

    // Create a list component and place two entries in the list
    lsMain = new List("List", Choice.IMPLICIT);
    lsMain.append("Coffee Beans", null);
    lsMain.append("Chocolate", null);

    // Add the command onto the list
    lsMain.addCommand(cmExit);

    // Listen for events
    lsMain.setCommandListener(this);
  }

  // Called by application manager to start the MIDlet.
  public void startApp()
  {
    if (display == null)
      display = Display.getDisplay(this);
    display.setCurrent(lsMain);
  }

  // A required method
  public void pauseApp()
  { }

  // A required method
  public void destroyApp(boolean unconditional)
  { }

  // Check to see if our Exit command was selected
  public void commandAction(Command c, Displayable s)
  {
    if (c == cmExit)
    {
      destroyApp(false);
      notifyDestroyed();
    }
  }
}
```

Compile and pre-verify

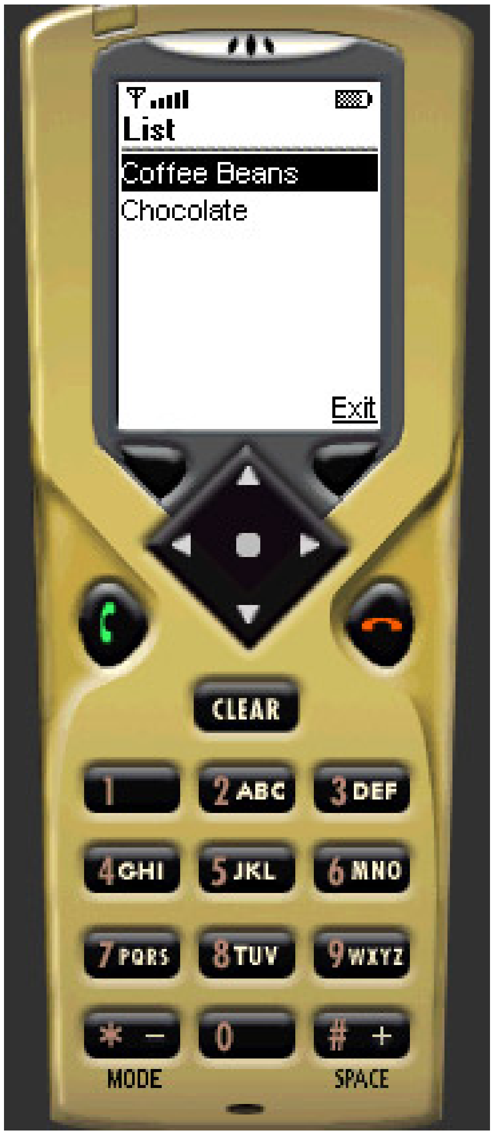
From a command prompt, enter the following to compile, pre-verify, and run the MIDlet:

javac -bootclasspath c:\j2me\midp1.0.3fcs\classes SecondMIDLet.java

preverify -classpath c:\j2me\midp1.0.3fcs\classes;. -d . SecondMIDlet

midp -classpath . SecondMIDlet

This figure shows the List component displayed on the emulator.



Create the JAR

Before packaging our files into a JAR, we'll have to create a manifest file. In the same directory as the source files, create a file with the name ***manifest.txt*** and enter/save the following:

```
MIDlet-Name: MIDlet Examples
MIDlet-Version: 1.0
MIDlet-Vendor: Core J2ME Technology
MIDlet-1: First MIDlet, /MIDlet1.png, FirstMIDlet
MIDlet-2: Second MIDlet, /MIDlet2.png, SecondMIDlet
MicroEdition-Profile: MIDP-1.0
```

```
MicroEdition-Configuration: CLDC-1.0
```

Create the JAR file by entering the following from the command prompt (although it may appear wrapped on your display/printout, the entire text must be on one line):

jar cvfm MIDlets.jar manifest.txt FirstMIDlet.class SecondMIDlet.class MIDlet1.png MIDlet2.png

This will create a JAR file with the name MIDlets.jar. It will contain a manifest file with the contents obtained from manifest.txt, and will also incorporate the class and PNG image files shown.

Note: The images MIDlet1.png and MIDlet2.png are optional. The images are included in the source code download for this tutorial. If you would prefer to create the suite without the images, go inside manifest.txt and change the reference to the MIDlet attributes:

```
MIDlet-1: First MIDlet, ,FirstMIDlet
MIDlet-2: Second MIDlet, ,SecondMIDlet
```

Any white space between attribute parameters is ignored.

Create the JAD

In the same directory as the source files, create a file with the name ***MIDlets.jad*** and enter/save the following:

```
MIDlet-Name: MIDlet Examples
MIDlet-Version: 1.0
MIDlet-Vendor: Core J2ME Technology
MIDlet-Description: Basic MIDlets
MIDlet-Jar-URL: MIDlets.jar
MIDlet-Jar-Size: 3188
MIDlet-1: First MIDlet, /MIDlet1.png, FirstMIDlet
MIDlet-2: Second MIDlet, /MIDlet2.png, SecondMIDlet
```

Note: If you are not using the images, change the last two lines of the JAD:

```
MIDlet-1: First MIDlet, ,FirstMIDlet
```

```
MIDlet-2: Second MIDlet, ,SecondMIDlet
```

Load the MIDlet suite

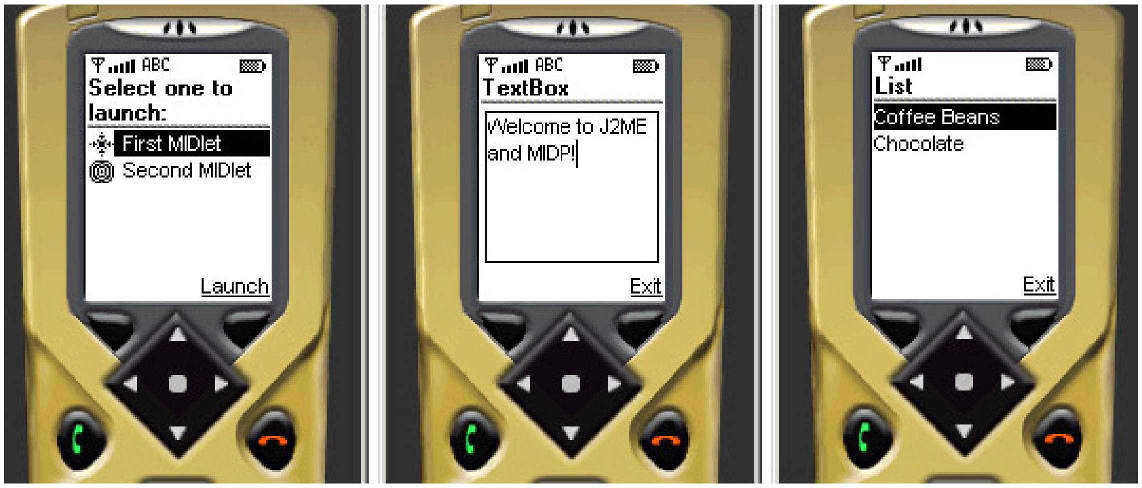
Here is the command to load the MIDlet suite onto the mobile device emulator:

midp -classpath MIDlets.jar -Xdescriptor MIDlets.jad

In the leftmost screen shot below, take notice of how the application manager presents a menu to choose between the two MIDlets within the suite.

Each entry in the menu -- the image and MIDlet name -- are obtained from the following lines in the manifest/JAD:

```
MIDlet-1: First MIDlet, /MIDlet1.png, FirstMIDlet
MIDlet-2: Second MIDlet, /MIDlet2.png, SecondMIDlet
```



Once an entry in the menu is selected, the application manager starts the MIDlet by loading the appropriate class file, the last parameter in the lines shown above.

Section 7. Parting shots

Keeping class files separate

Previously, we issued the following commands to compile and pre-verify our MIDlets:

```
javac -bootclasspath c:\j2me\midp1.0.3fcs\classes FirstMIDlet.java
```

```
preverify -classpath c:\j2me\midp1.0.3fcs\classes;. -d . FirstMIDlet
```

With this approach, the pre-verified class file overwrote the class file created by the compiler. For small applications, this is not much of an issue. However, as the complexity of your applications increase, so will the number of files that you need to juggle. One easy way to lend some organization is to place class files from the compiler and pre-verifier into separate directories.

Steps to separate class files:

#1 - Create new directories

Create directories with the names *jclasses* and *pclasses*.

These directories will hold the output from the compiler and pre-verifier, respectively.

#2 - Change command line syntax for compiler and pre-verifier

```
javac -bootclasspath c:\j2me\midp1.0.3fcs\classes -d jclasses *.java
```

```
preverify -classpath c:\j2me\midp1.0.3fcs\classes -d pclasses jclasses
```

-d jclasses informs the Java compiler to write the class files into the directory jclasses.

-d pclasses jclasses informs the pre-verifer to write the pre-verified class files into the directory pclasses. The reference to jclasses instructs the pre-verifier where to locate the class files to pre-verify.

#3 - Change command line for creating the JAR

```
jar cvfm MIDlets.jar manifest.txt -C pclasses . MIDlet1.png MIDlet2.png
```

-C pclasses . informs the JAR program to look in the directory pclasses and archive all (".") the files.

#4 - Load MIDlet suite

```
midp -classpath MIDlets.jar -Xdescriptor MIDlets.jad
```

J2ME Wireless Toolkit

Sun Microsystems provides a free J2ME Wireless Toolkit to simplify the development cycle when writing MIDlets. The toolkit manages project directories, compiles, pre-verifies, and packages MIDlets (creates the JAR and JAD). The toolkit also provides several emulators to preview MIDlets.

Download J2ME Wireless Toolkit at http://java.sun.com/products/j2mewtoolkit.

Resources and additional information

- Download the *source code* for examples in this tutorial.
- You can find additional articles and resources at *Core J2ME* (http://www.CoreJ2ME.com).
- Find open enrollment, onsite, and Web-based J2ME training at *http://www.TheWirelessMind.com*.
- Read about *VisualAge Micro Edition* supporting CLDC and MIDP.
- Visit the *developerWorks Wireless zone*, a valuable resource for information on wireless technology and development.
- Read the *developerWorks* article "*Think Small with J2ME*."
- Get more information on the *Java Development Kit 1.3*(http://java.sun.com/products/jdk/1.3).
- Learn about *MIDP software*(http://java.sun.com/products/midp).
- Read more about *CLDC software* (http://java.sun.com/products/cldc).
- Get information on the *J2ME Wireless Toolkit*(http://java.sun.com/products/j2mewtoolkit).
- Read Sun's *JAR file tutorial* (http://java.sun.com/docs/tutorial/jad/index.html).
- *Java Community Process*(http://www.jcp.org).
- *Java Specification Request for Mobile Information Device Profile* (http://www.jcp.org/jsr/detail/37.jsp).
- *Java Specification Request for PDA Profile* (http://www.jcp.org/jsr/detail/75.jsp).
- Learn more about the *PNG image file format*(http://java.sun.com/products/midp).

Section 8. Feedback

Feedback

Please send us your feedback on this tutorial. We look forward to hearing from you!

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.