User's Guide

Wireless Toolkit Version 1.0.4

Java™ 2 Platform, Micro Edition

Please

Adobe PostScript™

Contents

Figures

Tables

Preface

The *Java*^{TM} *2 Platform, Micro Edition, Wireless Toolkit User's Guide* describes how to install, configure and work with the J2ME^{TM} Wireless Toolkit and its components.

Who Should Use This Book

This guide is intended for developers creating MIDP applications with the J2ME Wireless Toolkit. This document assumes that you are familiar with Java programming, Mobile Information Device Profile(MIDP), and the Connected Limited Device Configuration (CLDC).

How This Book Is Organized

This guide contains the following chapters and appendixes:

Chapter 1 introduces the J2ME Wireless Toolkit and the MIDlet development features it provides.

Chapter 2 describes system requirements and installation instructions for the J2ME Wireless Toolkit. Also included is a information on configuring the PalmOS emulator for use with the Wireless Toolkit.

Chapter 3 explains how to perform basic programming operations with KToolBar, such compiling, preverifying, debugging, tracing, and packaging.

Chapter 4 describes the performance tuning features: profiling, memory monitoring, network monitoring, and speed emulation.

Chapter 5 describes the example devices and demo applications provided by the Wireless Toolkit. This chapter also explains how to input text to the devices, how to set device preferences, and how to access the device utilities.

Chapter 6 describes command line operations, including arguments and options, available in the J2ME Wireless Toolkit. This chapter includes an example of stepping through a basic development cycle working from the command line.

Chapter 7 describes how to test and demonstrate the over the air initiated provisionig process.

Appendix A lists and describes MIDlet attributes.

Appendix B describes how to demonstrate MIDlet s for non-development purposes.

Appendix C describes internationalization features in the Wireless Toolkit.

Appendix D explains how certificate authority managers are used in the Wireless Toolkit.

Using Operating System Commands

This document may not contain information on basic UNIX® or Microsoft Windows commands and procedures such as opening a terminal window, changing directories, and setting environment variables. See the software documentation that you received with your system for this information.

Typographic Conventions

Typeface	Meaning	Examples
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail.`
`AaBbCc123`	What you type, when contrasted with on-screen computer output	`%` **`su`** `Password:`
AaBbCc123	Book titles, new words or terms, words to be emphasized	Read Chapter 6 in the *User's Guide*. These are called *class* options. You *must* be superuser to do this.
	Command-line variable; replace with a real name or value	To delete a file, type `rm` *filename*.

Shell Prompts

Shell	Prompt
C shell	*machine_name*`%`
Microsoft Windows	*\<directory\>*

Related Documentation

Application	Title
Customization	*J2ME Wireless Toolkit Basic Customization Guide*
MIDP	*Building and Running MIDP*

Accessing Sun Documentation Online

The Java Developer Connection^{sm} web site enables you to access Java™ platform technical documentation on the Web:

http://developer.java.sun.com/developer/infodocs/

Sun Welcomes Your Comments

We are interested in improving our documentation and welcome your comments and suggestions. You can email your comments to us at:

j2mewtk-comments@sun.com

Introduction to the Wireless Toolkit

The *Java*^{TM} *2 Platform, Micro Edition, Wireless Toolkit User's Guide* describes how to install, configure and work with the J2ME^{TM} Wireless Toolkit and its components.

The J2ME Wireless Toolkit supports the development of Java applications that run on devices compliant with the Mobile Information Device Profile (MIDP), such as cellular phones, two-way pagers, and palmtops.

This document assumes that you are familiar with Java programming, MIDP, and the Connected Limited Device Configuration (CLDC). You can find more information about MIDP and CLDC at the following URLs:

- `http://java.sun.com/products/midp`
- `http://java.sun.com/products/cldc`

Overview

The J2ME Wireless Toolkit supports a number of ways to develop MIDP applications. You can carry out the development process by running the tools from the command line or by using development environments that automate a large part of this process.

The KToolBar, included with the J2ME Wireless Toolkit, is a minimal development environment with a GUI for compiling, packaging, and executing MIDP applications. The only other tools you need are a third-party editor for your Java source files and a debugger. For more information on the KToolbar, see Chapter 3, "Operating with KToolBar."

An IDE compatible with the J2ME Wireless Toolkit provides even more convenience. For example, when you use the Sun ONE Studio 4, Mobile Edition, (formerly Forte^{TM} for Java^{TM}) you can edit, compile, package, and execute or debug MIDP applications, all within the same environment.

For a list of other IDEs that are compatible with the Wireless ToolKit, see `http://java.sun.com/products/j2mewtoolkit/`, the Java^{TM} 2 Platform Micro Edition, Wireless Toolkit web page.

This section describes the phases of MIDP application development outside of editing, and how the toolkit contributes to these phases. The phases are illustrated in the following diagrams.
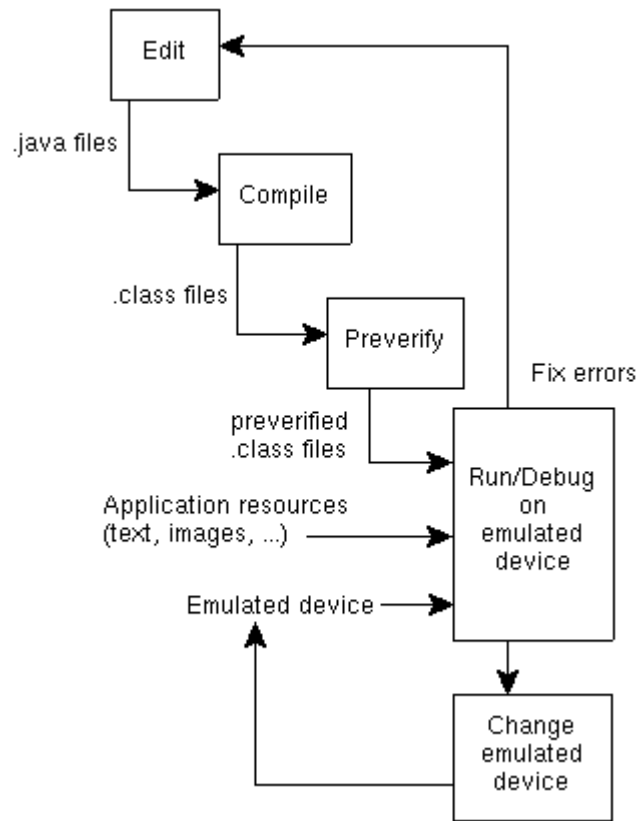
FIGURE 1   Developing and Testing an Application
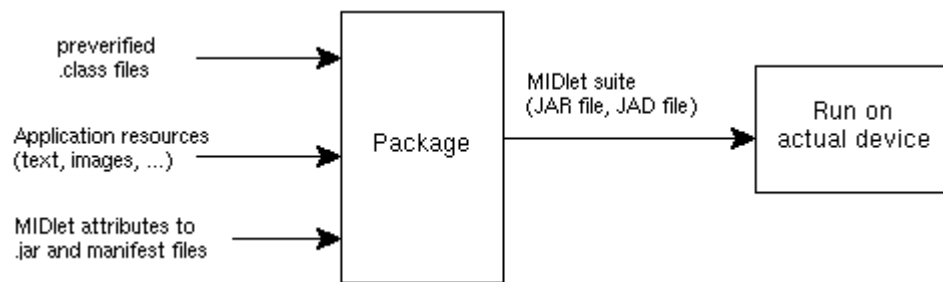
FIGURE 2   Packaging an Application

Compilation and Prefabrication

When you use KToolbar or a toolkit-compatible environment, such as the Sun ONE Studio 4, Mobile Edition, the environment compiles your source files for you, using the Java 2 SDK, Standard Edition (J2SE^{TM} SDK) compiler.

After compiling the sources, the development environment passes the generated class files to the Preverifier. This tool rearranges bytecodes in the classes to simplify the final stage of bytecode verification on the CLDC virtual machine. It also checks for the use of virtual machine features that are not supported by the CLDC.

For more information on how to compile and preverify files using KToolbar, see Chapter 3, "Operating with KToolBar."

You can also perform compilation and preverification by running the tools on the command line. For more information, see Chapter 6, "Operating From the Command Line."

Running and Debugging

When you use KToolbar or a toolkit-compatible environment such as the Sun ONE Studio 4, Mobile Edition, you can run and debug applications within the environment using the Emulator, which simulates the execution of the application on different target devices.

The Emulator enables you to approximate the experience a user has with an application on a particular device, and to test the portability of the application across different devices. For more information, see Chapter 5, "Working With the Emulator."

You can also open the Emulator from the command line. For more information, see Chapter 6, "Operating From the Command Line."

Packaging

MIDP applications, or MIDlets, are packaged into a MIDlet suite, a grouping of MIDlets that can share resources at runtime. The following diagram illustrates how a MIDlet suite is organized.
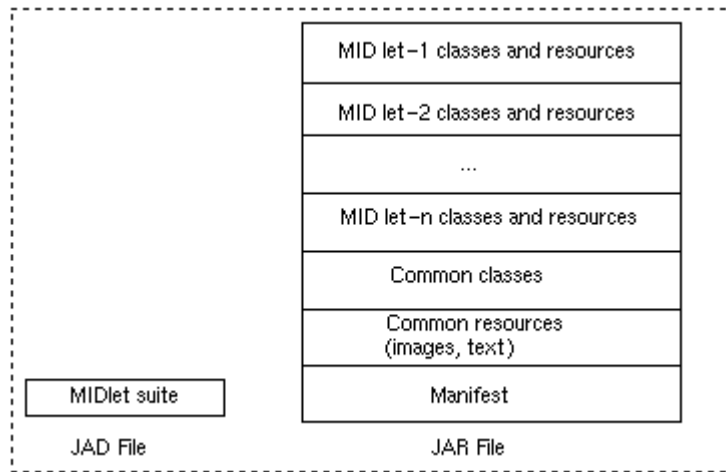
FIGURE 3   MIDlet Suite Components

More formally, a MIDlet suite includes:

- **A Java Application Descriptor (JAD) file.** This file contains a predefined set of attributes (denoted by names that begin with "MIDlet-") that allow application management software to identify, retrieve, and install the MIDlets. All attributes appearing in the JAD file are made available to the MIDlets. You can define your own application-specific attributes and add them to the JAD file. (For more information on what attributes go into the JAD file, see Appendix A, "MIDlet Attributes.")

- **A Java Archive (JAR) file.** The JAR file contains:
  - Java classes for each MIDlet in the suite.
  - Java classes shared between MIDlets.
  - Resource files used by the MIDlets (for example, image files).
  - A manifest file describing the JAR contents and specifying attributes used by application management software to identify and install the MIDlet suite. (For more information on what attributes go into the manifest, see Appendix A, "MIDlet Attributes.").

Development environments such as KToolBar and the Sun ONE Studio 4, Mobile Edition automate the packaging of MIDlet suites. (For more information on how to package applications using KToolBar, see "Packaging" on page 21.) To package MIDlet suites from the command line, you need the J2SE SDK JAR tool to create JAR files, and a text editor for creating JAD files.

Packaging Obfuscated Source Code

An additional feature of the J2ME Wireless Toolkit is the ability to build an obfuscated package. You are required to obtain a code obfuscator plug-in to use this feature. The JAR file for the code obfuscator should be placed in the *j2mewtk.dir*\bin directory.

Note – Creating an obfuscated package is only available through the KToolBar.

Obfuscation removes extraneous class information, such as local variable names. Classes, methods, interfaces, and such are renamed so as to make them ambiguous. An obfuscated package protects your project files from decompilation and reverse engineering. In addition to protecting your source code, the obfuscation process reduces the size of the classes resulting in smaller JAR files. The details of how code is obfuscated is dependent on the specific code obfuscator you choose to use.

When creating an obfuscated package, preverification is done after the code has been obfuscated rather than immediately after compilation.

For information on creating packages and obfuscated packages, see "Packaging" on page 21 in Chapter 3, "Operating with KToolBar."

Installing the Wireless Toolkit

This chapter describes the system requirements for the J2ME Wireless Toolkit and how to install the toolkit.

System Requirements

You can use the J2ME Wireless Toolkit on Windows NT 4.0, or Windows 2000 operating systems. (Unsupported versions of the toolkit for Solaris and Linux are also available.) The toolkit requires 30MB of hard disk space, and 64MB of memory at runtime.

Also, the Java 2 Software Development Kit, Standard Edition 1.3 or higher must already be installed. You can download the J2SE SDK from `http://java.sun.com/j2se/1.4/`.

If you want to run the tools with the Palm OS Emulator(POSE), you should install the emulator before installing the toolkit. You can download the emulator from `http://www.palmos.com/dev/tech/tools/emulator/`. Be sure to follow the POSE installation instructions.

For information on the system requirements for the Palm OS Emulator, refer to its documentation. For more information on the J2ME Wireless Toolkit, including other IDEs that support the toolkit, see the page `http://java.sun.com/products/j2mewtoolkit`, for the Java 2 Platform Micro Edition, Wireless Toolkit.

Installation Procedure

To install the toolkit, run the installer, `j2me_wireless_toolkit-1_0_4.exe`. (If you downloaded this file instead of running it directly from the Web site, double-click the installer's icon.) Follow the instructions the installer provides.

Note – During the installation you are asked for the directory in which to place the toolkit. This path should not contain spaces; if it does, some of the tools will not work. For example, the directory `C:\Program Files\wtk1.0.4` is not desirable because there is a space between `Program` and `Files`.

The following table shows the contents of the installation directory, which is referred to as {*j2mewtk.dir*} throughout this guide.

TABLE 1   J2ME Wireless Toolkit Directory Contents

File or Directory	Description
`BinaryLicense.html`	License agreement.
`BinaryReleaseNotes.html`	Release notes.
`index.html`	Index pointing to the toolkit documentation.
`appdb\`	Directory containing database files, such as RMS files and ME keystore files.
`apps\`	Directory containing demo applications and additional applications created by the KToolBar.
`bin\`	Batch files and executables for the tools.
`docs\`	Directory containing user and API documentation, including this guide.
`lib\midpapi.zip`	Archive containing the CLDC and MIDP API classes. These files are used during the compilation of the application source files and the byte-code pre-verification of the application classes.
`sessions\`	Default directory containing profiling, memory monitoring, and network monitoring session files.
`wktlib\devices\`	Directory containing property files for devices emulated by the Emulator.

The installation also creates a program group for the J2ME Wireless Toolkit.

Configuring the Palm OS Emulator

If you want to use the Palm OS Emulator with the J2ME Wireless Toolkit, you must first configure it as follows:

1. **Redirect NetLib calls to the host's TCP/IP.**

   For certain functions like debugging and Internet connectivity to work, you must set NetLib API calls to be redirected from POSE to use your computer's TCP/IP.

   a. **Run POSE and right-click the emulator.**

      A menu appears.

   b. **Select Settings -> Properties...**

      The Properties dialog appears.
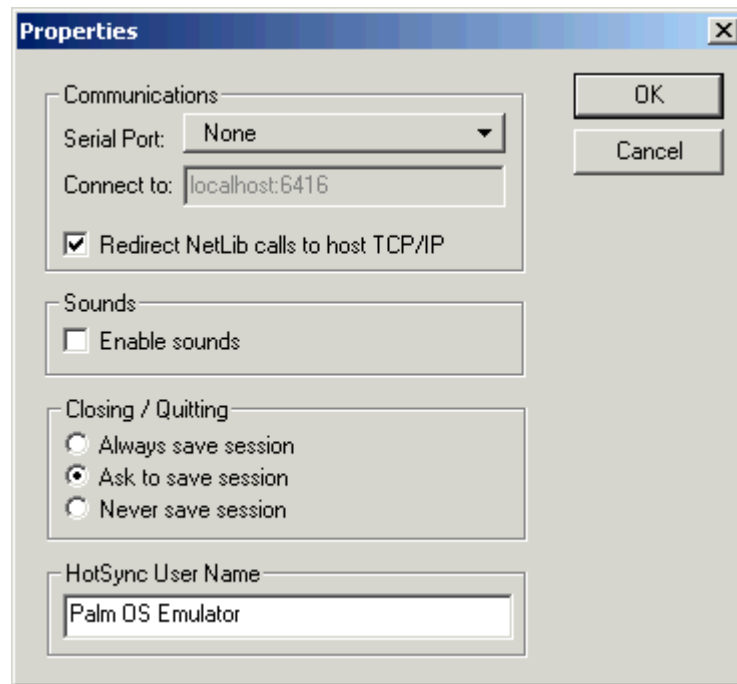
FIGURE 4   Properties Dialog

   c. **Check the Redirect NetLib calls to host TCP/IP box, and click OK.**

      The Properties dialog disappears.

   d. **Right-click the emulator and select Save.**

      Your changes are saved.

2. **Disable debugging.**

   POSE allows various items to be debugged while the application executes. However, for POSE to work with the J2ME Wireless Toolkit, debugging must be disabled altogether.

   a. **Run POSE, and right-click the emulator.**

      A menu appears.

b. Select Settings -> **Debugging...**

The Debug Options dialog appears.
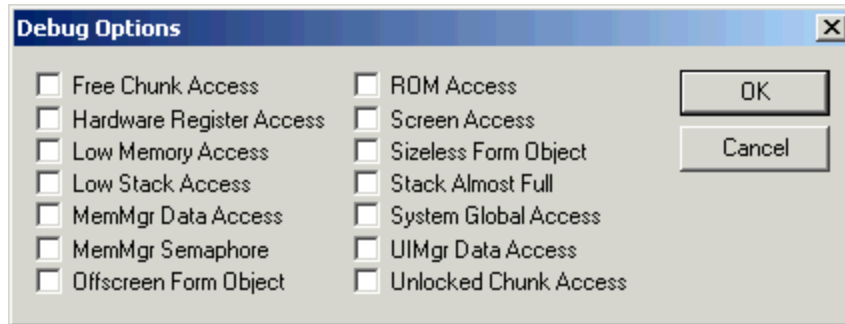
FIGURE 5 Debug Options Dialog

c. Uncheck all the boxes and click OK.

The Debug Options dialog disappears.

d. Right-click the emulator and select Save.

Your changes are saved.

3. Set the POSE location in the J2ME Wireless Toolkit.

The first time you run an application using POSE through the J2ME Wireless Toolkit, a dialog appears, asking you for the POSE location. After you set the location, the dialog does not appear again when you run MIDP applications using the emulator.
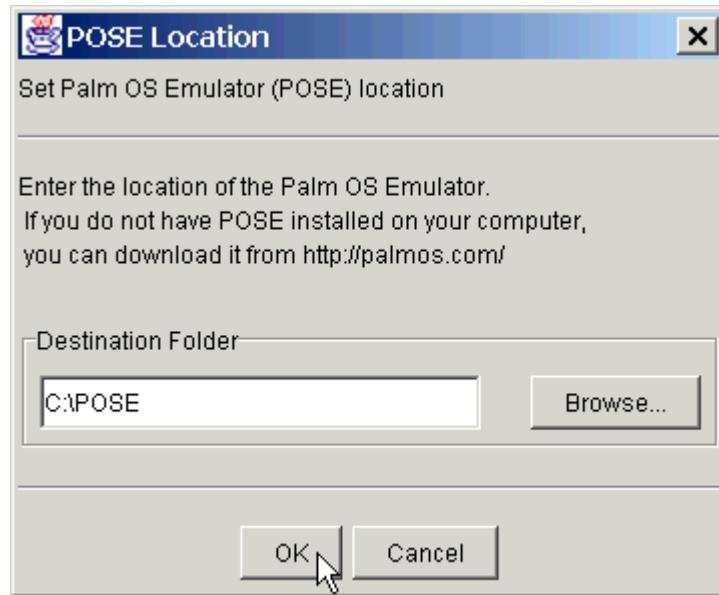
FIGURE 6   POSE Location Dialog

If you need to change the POSE location after you have first set it, use the Palm OS Preferences dialog. For more information, see "Configuring the Palm OS Emulator" on page 8.

Operating with KToolBar

KToolBar is a minimal development environment for developing MIDlet suites. From the KToolBar, you can:

- Create a new project or open an existing one
- Build, run, and debug your MIDlet
- Fine tune your MIDlet application
- Package your project files
- Modify the attributes of your MIDlet suite

To run the KToolBar:

- **Choose Programs -> J2ME Wireless Toolkit 1.0.4 -> KToolbar from the Microsoft Windows Start menu.**

    The main window appears:
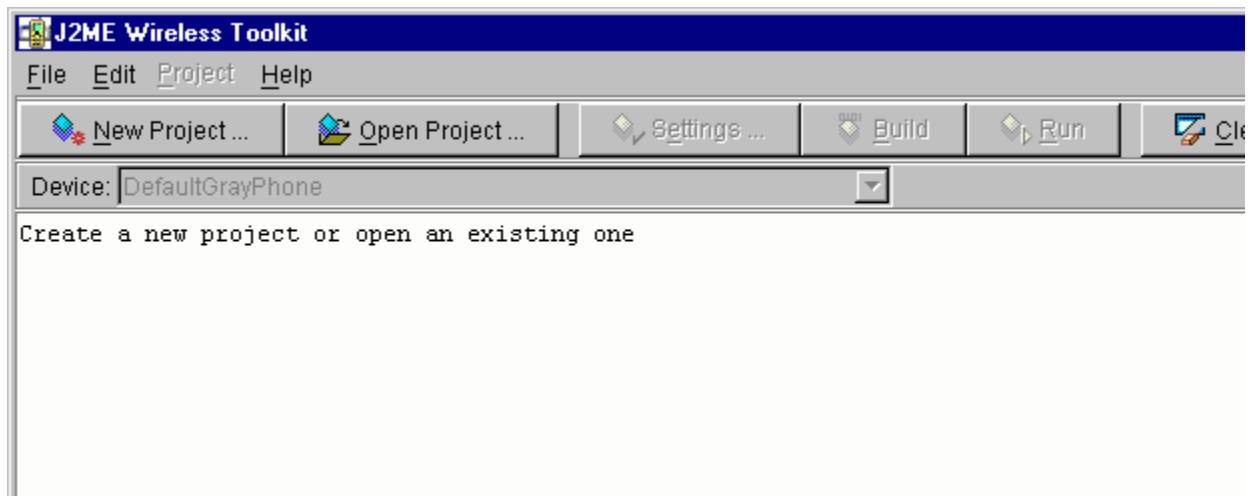
FIGURE 7   KToolBar Main Window

Navigating in KToolBar

You can navigate through KToolBar windows (the main window, Profiling, Memory Monitor, and Network Monitor windows) using the Tab and arrow keys. Mnemonics on menus and buttons provide you with alternative means to initiating commands. A mnemonic is the underlined letter that corresponds to the keyboard key to press in conjunction with the Alt key to activate a command or to navigate to a component in the window.

You can press the Tab key to bring the focus to a particular component of a window and then use the arrow keys to manipulate that component.

KToolBar Projects

A KToolBar project is associated with a MIDlet suite. The project contains the suite's source, resource and binary files, as well as the JAD and manifest files that contain the suite's attributes.

Project files are located in project subdirectories under the Wireless Toolkit's installation directory, {*j2mewtk.dir*}. The following table shows how files are organized within the directory for the project, {*project.name*}:

TABLE 2   Project File Organization

Directory	Description
{*j2mewtk.dir*}\apps\{*project.name*}	Contains all source, resource, and binary files of the project
{*j2mewtk.dir*}\apps\{*project.name*}\src	Contains all the source files.
{*j2mewtk.dir*}\apps\{*project.name*}\res	Contains all the resource files.
{*j2mewtk.dir*}\apps\{*project.name*}\bin	Contains the JAR, JAD, and unpacked manifest files.
{*j2mewtk.dir*}\apps\{*project.name*}\lib	Contains external class libraries, in JAR or ZIP format for a specific project.
{*j2mewtk.dir*}\apps\lib	Contains external class libraries, in JAR or ZIP format for all KToolBar projects.

Note – Adding external class libraries to a project increases the size of the MIDlet suite's JAR file. Large JAR files take longer to load onto a device, and might be unusable on devices with low memory.

Creating a New Project

To create a new project:

1. **Choose File -> New Project from the menu or click New Project on the toolbar.**

   The New Project dialog appears.

2. **Type the name of the project in the Project Name field, and the name of the main MIDlet class in the MIDlet Class Name field.**

   For example, you might call the project newproject, and the MIDlet class might be myTest.Hello.

3. **Click Create Project.**

   The main window's title changes to include the name of the new project, as shown by the following screenshot.

   The console indicates where to place your source, resource, and library files. The locations are consistent with the project file organization outlined in TABLE 2 on page 14.
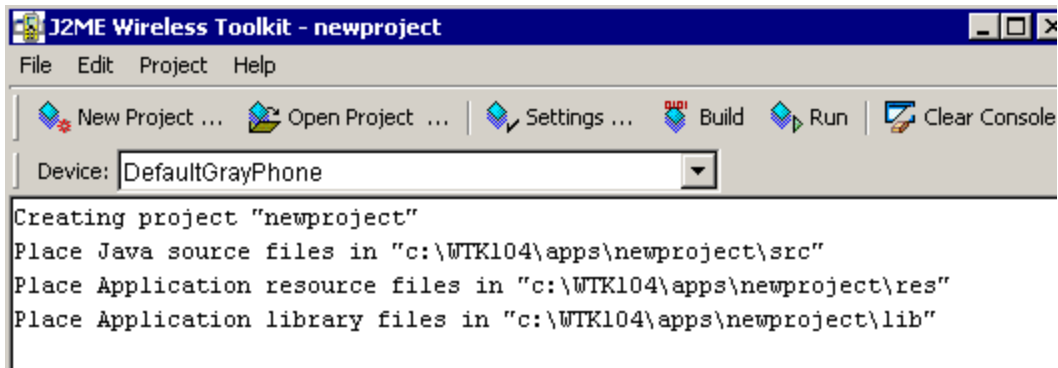
FIGURE 8 Console Output After Creating a Project

Opening an Existing Project

To open an existing project:

1. **Choose File -> Open Project from the menu or click Open Project on the toolbar.**

   The Open Project dialog appears with a list of projects.

2. **Double-click the project, or choose the project and click Open Project.**

   The main window's title changes to include the name of the project.

Editing MIDlet Suite Attributes

To edit a MIDlet suite's attributes, use the project settings dialog, as shown in the screenshot below. To bring up this dialog, choose Project -> Settings from the menu or click the Settings button on the toolbar.
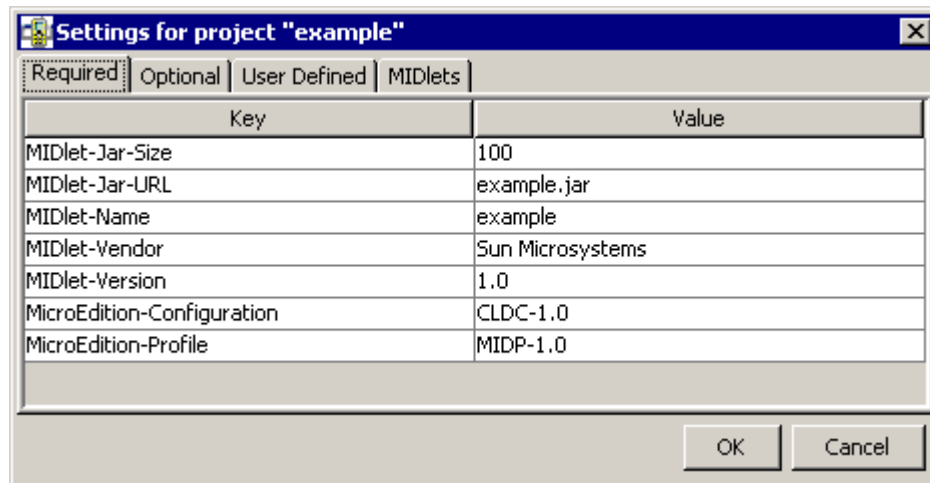
FIGURE 9   Project Settings Dialog

This section explains how to use the project settings dialog to modify a MIDlet suite's attributes. For more information about the attributes themselves, see Appendix A, "MIDlet Attributes."

Modifying MIDlet Suite Attributes

To modify a required, optional, or user-defined MIDlet suite attribute:

1. **Click the Required, Optional, or User Defined tab.**

   Depending on which tab you click, the Required, Optional, or User Defined pane appears.

2. **Choose the attribute, and click on its value field.**

   A cursor appears in the value field.

3. **Make your changes, and press Enter.**

   The attribute's entry reflects your changes.

4. **Click OK to save your changes.**

Modifying MIDlet-Specific Attributes

To modify an individual MIDlet's name, icon, and class:

1. **Click the MIDlets tab.**

   The MIDlets pane appears.

2. **choose the MIDlet, and click Edit.**

   The Enter MIDlet Details dialog appears.

3. **Make your changes, and click OK.**

   The Enter MIDlet Details dialog disappears, and the MIDlet's entry reflects your changes.

4. **Click OK to save your changes.**

Adding User-Defined Attributes

To add a user-defined attribute:

1. **Click the User Defined tab.**

   The User Defined pane appears.

2. **Click Add.**

   The Add Property dialog appears.

3. **Enter the name of the attribute, and click OK.**

   The Add Property dialog disappears, and a new entry is created for the attribute.

4. **Choose the attribute, and click on its value field.**

   A cursor appears in the value field.

5. **Make your changes, and press Enter.**

   The attribute's entry reflects your changes.

6. **Click OK to save your changes.**

> **Note –** Don't use the prefix "MIDlet-" for a user-defined attribute. This format is reserved for system-defined MIDlet attributes.

Removing User-Defined Attributes

To remove a user-defined attribute:

1. **Click the User Defined tab.**

   The User Defined pane appears.

2. **Select the attribute, and click Remove.**

   KToolbar asks if you are sure. If you are, then press Yes.

3. **Click OK to save your changes.**

Adding MIDlet-Specific Attributes

To add a MIDlet-specific attribute:

1. **Click the MIDlets tab.**

   The MIDlets pane appears.

2. **Click Add.**

   The Enter MIDlet Details dialog appears.

3. **Enter the MIDlet's attributes, and click OK.**

   The Enter MIDlet Details dialog disappears, and a new entry is created for the MIDlet.

4. **Click OK to save your changes.**

Removing MIDlet-Specific Attributes

To remove a set of MIDlet-n attributes specific to a particular MIDlet:

1. **Click the MIDlets tab.**

   The MIDlets pane appears.

2. **Select the MIDlet, and click Remove.**

   KToolbar asks if you are sure. If you are, then press Yes.

3. **Click OK to save your changes.**

Changing the Order of the MIDlets

To change the order of the MIDlets in the suite (that is, the order in which they are listed when you launch the suite):

1. **Click the MIDlets tab.**

   The MIDlets pane appears.

2. **Select a MIDlet to move, and click Move Up or Move Down.**

   When you move the MIDlet, its number in the sequence is updated automatically.

3. **Click OK to save your changes.**

Compiling and Preverifying

The KToolBar compiles and preverifies source code in one sequence. To compile and preverify your source code:

● **Choose Project ->Build or click Build on the toolbar.**

The sources are compiled against the MIDP and CLDC APIs, as well as any libraries in the project's `lib\` folder and the toolkit's `\apps\lib\` folder.

Note – The Java classes are compiled with debugging information. In the packaging stage the Java classes are compiled without debugging information.

Running

To run the current MIDlet suite in the Emulator using the KToolbar:

1. **(Optional) Use the Device menu to select the device to be emulated.**

   The list displays the devices available for the loaded application.

2. **Choose Project -> Run or click Run on the toolbar.**

   The Emulator appears, running your MIDlet suite.

   The console displays system and trace output as a MIDlet suite executes.

   To clear the console:

- **Choose Edit -> Clear Console from the menu or click Clear Console on the toolbar.**

    (To specify what actually gets outputted to the console, see "Enabling Tracing" on page 54 in Chapter 5, "Working With the Emulator.")

Debugging

You can debug an application from within KToolBar by connecting to remote debugging facilities, such as an IDE.

To debug an application under KToolBar:

1. **Choose Project -> Debug.**

    The dialog asks you to enter a TCP/IP port number which the external debugger can use to connect to the emulator.

2. **Enter a TCP/IP port.**

    In most cases you can use the default value, but you should use another value if another application is using this port, or if you encounter problems connecting to the emulator from the debugger.

3. **Click Debug.**

    The emulator begins running in debugging mode, and waits for a connection from a debugger.

4. **Start the remote debugger and attach it to the TCP/IP port you specified in step 2.**

    Make sure to set the remote debugger to run in remote mode and to use TCP/IP. For more information, consult the debugger's documentation.

Cleaning Up Project Files

To remove obsolete or unnecessary files in your project directory:

- **Choose Project -> Clean.**

    The Clean command deletes all temporary and class files in the current project directory.

Packaging

You can create a package of your project files or create an obfuscated package to protect your code from possible decompilation. Another benefit to creating an obfuscated package is that the obfuscation process reduces the size of the Java bytecode, resulting in a smaller JAR file and possibly faster download times.
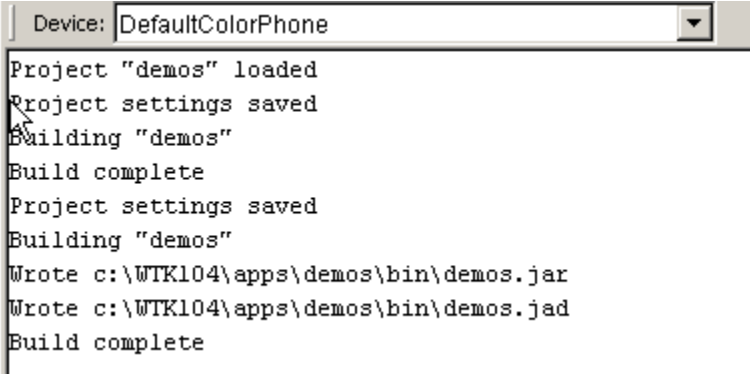
To build a package:

- **Choose Project** -> **Package** -> **Create Package or Create Obfuscated Package.**

Choosing Create Package creates a standard `.jar` file. When the classes are packaged, they are compiled without debugging information to reduce the size of the JAR file.

Choosing Create Obfuscated Package creates a `.jar` file containing obfuscated code making it difficult to reverse engineer. Specifically how the contents of your package are obfuscated is dependent on the type of obfuscation tool you choose to use. To use this feature, you must already have a code obfuscator plug-in.

A progress bar appears when packaging begins. When the packaging finishes, the output display indicates where the JAR and JAD files have been placed.

FIGURE 10   Console Output After Packaging

Implementing Support for Code Obfuscation

The J2ME Wireless Toolkit contains a support framework for byte code obfuscators. It also contains a plug-in for the RetroGuard byte code obfuscator. You can obtain the `retroguard.jar` file from `http://www.retrologic.com`. The `retroguard.jar` file must be placed in the Wireless Toolkit's `bin` directory: {*j2mewtk.dir*}`\bin`.

If you choose to use a code obfuscator other than RetroGuard, you must implement the plug-in yourself. See the *Wireless Toolkit Basic Customization Guide* for an example of how to implement a code obfuscator plug-in.

Using Class Libraries

KToolBar enables you to build projects from source and resource files. Sometimes you want to use a class library for which you do not have source files. This section shows you how to build a project using an external class library.

You should be cautious when including external class libraries. Adding unnecessary class libraries to a project increases both the time needed to package it and the size of the resulting MIDlet suite JAR file. A large JAR file increases the time needed to load the MIDlet suite, and could prevent it from running on devices with low memory.

Class libraries for use with KToolBar should be compatible with the CLDC and MIDP APIs and should be packaged in `.jar` or `.zip` format. KToolBar provides ways for you to develop with class libraries, both on a per project and on a global basis.

External Libraries for a Specific Project

To add class libraries to a KToolBar project, locate the directory containing your application (refer to TABLE 2 on page 14). The application's directory contains a subdirectory, `lib`. Place the JAR or ZIP file containing the class library into this subdirectory. For example, if you installed the J2ME Wireless ToolKit in `C:\WTK104` and your application is called `ExampleMIDlet`, the class library would go in the directory, `C:\WTK104\apps\ExampleMIDlet\lib`. When you build, run, debug, and package your project, the class files in the `lib` directory are used.

External Libraries for All Projects

You can also define class libraries to be available for all projects that you develop with KToolBar. To do this, place the JAR or ZIP files containing the classes in the subdirectory `apps\lib` of the directory in which you installed the J2ME Wireless ToolKit. For example, if you installed the Wireless Toolkit in `C:\WTK104`, you would place the class libraries in `C:\WTK104\apps\lib`. Class libraries in the `apps\lib` directory are used for all projects.

Note – Class libraries for a particular project can import classes and resources from any general library as well as specific libraries. Class libraries for projects in general can only import classes and resources from general class libraries.

Setting Emulator Preferences and Using Emulator Utilities

You can access the Emulator's Preferences and Utilities tools through the KToolBar menu.

To access the Emulator Preferences tool, choose Edit -> Preferences.
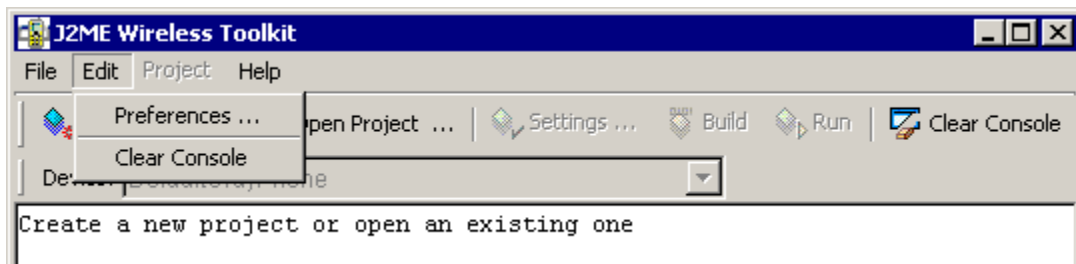
FIGURE 11   Accessing Emulator Preferences in KToolBar

To access the Emulator Utilities tool, choose File -> Utilities.
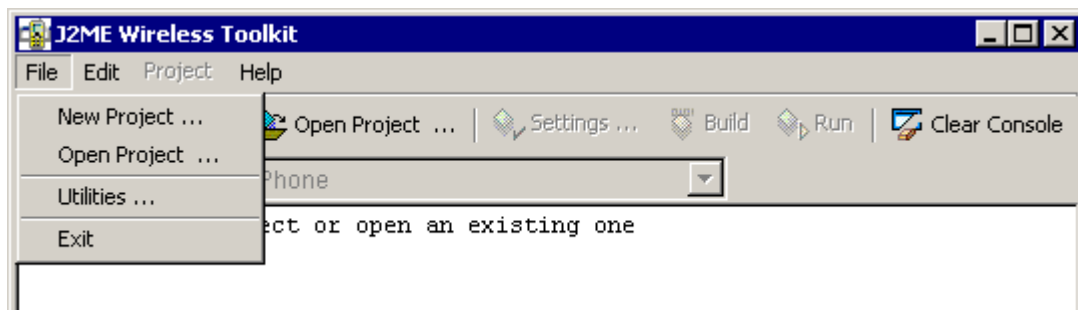
FIGURE 12   Accessing Emulator Utilities on KToolBar

For more information on using the Emulator Utilities and Preferences tools, see "Preferences and Utilities" on page 51 in Chapter 5, "Working With the Emulator."

Customizing KToolBar

KToolBar includes some advanced configuration options. You can use these options by editing the file {*j2mewtk.dir*}\wtklib\Windows\ktools.properties. To see the effects of your changes, restart KToolBar.

Setting the Application Directory

By default, the J2ME Wireless Toolkit stores MIDP applications in directories under {*j2mewtk.dir*}\apps. You can change this by adding a line to ktools.properties of the following form:

kvem.apps.dir: *<application_directory>*

Any backslash ('\') characters in the directory's path should be preceded by another backslash. Also, the directory's path should not contain any spaces.

For example, to set the application directory to D:\dev\midlets, you would use:

kvem.apps.dir: D:\\dev\\midlets

Setting the Javac Encoding Property

By default, the Java compiler uses the encoding set in the J2SE environment that you are running. For information on how to override the default source file encoding, see "Java Compiler Encoding Setting" on page 81 in Appendix C, "Internationalization."

Working with Revision Control Systems

Using the filterRevisionControl property, you can configure KToolBar to recognize and ignore auxiliary files created by the SCCS, RCS and CVS revision control systems. To do this, include the following line in ktools.properties:

kvem.filterRevisionControl: true

As a result, you prevent KToolBar from treating revision control files as source and resource files. For example, KToolBar would treat a file named src\SCCS\s.MyClass.java as being an SCCS revision control file and not a Java source file.

Performance Tuning Applications

You can examine various aspects of the MIDlet applications you created with the J2ME Wireless Toolkit to identify where you can improve the efficiency and speed of your MIDlet. The Wireless Toolkit enables you to optimize the performance of your MIDlet with the following features:

- **Profiler**. Enables you to examine the execution time and the frequency of use of the methods in your application.

- **Memory Monitor**. Enables you to examine memory usage in your application.

- **Network Monitor**. Lets you monitor transmissions between your device and the network.

- **Speed Emulation**. Enables you to adjust drawing speed to refine graphics rendering. It also enables you to adjust the speed of byte code execution and data transfer across the network to give you a sense of how quickly your application runs on a device.

Note – Turning on multiple performance features simultaneously can adversely affect the data collected by slowing down application execution. For more accurate results, try enabling one performance feature per data collection.

Profiling Your Application

You can examine the method execution time with the Profiler utility. The Profiler collects data from an emulator during runtime. By seeing how much time a method takes to execute, you can see where potential problems, such as bottlenecks, might exist in the application.

The Profiler window displays two types of method information:

- Method relationships shown in a hierarchical list called the Call Graph.

- Execution time and the number of times a method and its descendants were called during runtime.

Profiling Data Display

In the Call Graph tree, you see folders for top-level methods. Opening a method's folder displays the methods called by it. Selecting a method in the tree shows the profiling information for it and all the methods called by it. Selecting `<root>` displays profiling information for all methods in the program.
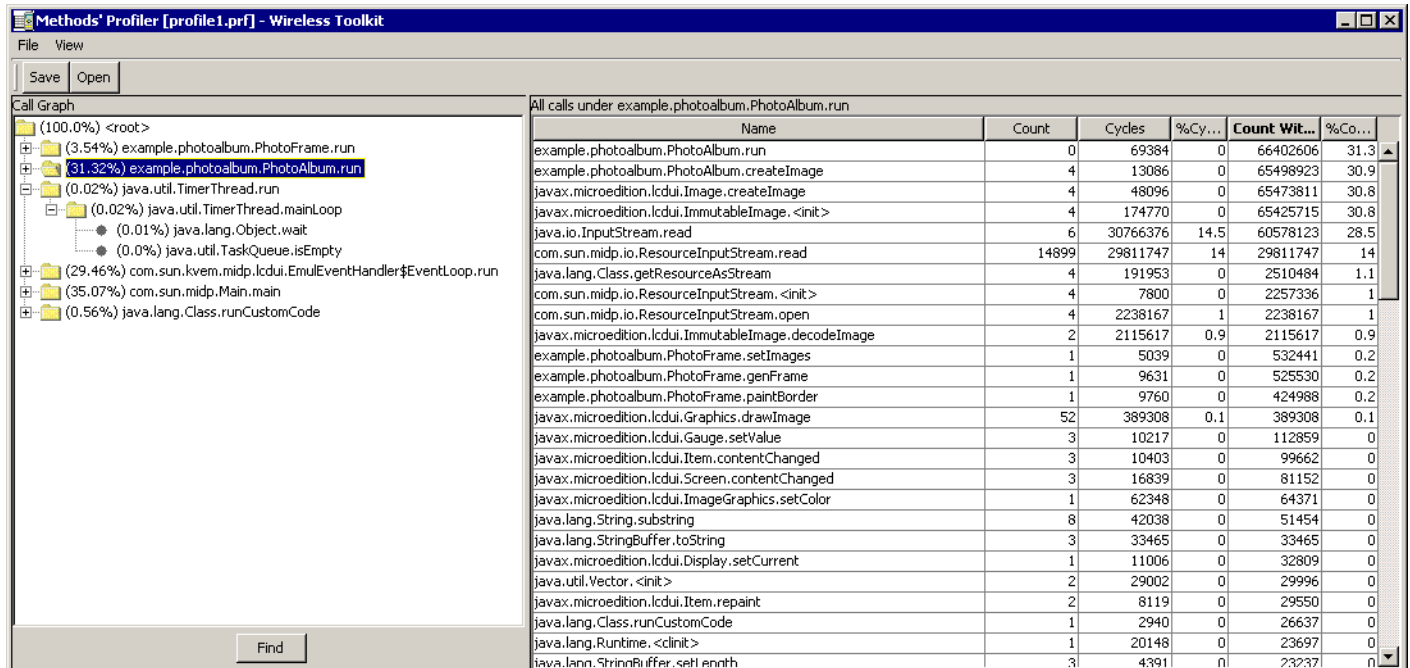
FIGURE 13   Profiler Window

The table displays rows of methods. For each method, you can see its:

- Name. The fully qualified name of the method.

- Count. The number of times the method was called during execution.

- Cycles. The execution time, in seconds, of a method (does not include the execution time of methods called by that method).

- %Cycles. The percentage of time spent on a method's execution in respect to the time the entire program ran (does not include the execution time of methods called by that method).

- Count With Children. The number of times the method and its descendants were called during execution.

- %Count With Children. The percentage time spent running the method and all of its descendants in respect to the time the entire program ran.

You can click on the column titles in the table to sort the display. Clicking on Name sorts the methods in alphabetical order. Clicking Count, Cycles,%Cycles, Count with Children, or %Count with Child sorts the information in ascending order. Clicking on the column title again resorts the information in descending order.

Viewing Profiling Information

To obtain profiling information, follow these steps:

1. **From the KToolBar, choose Edit** -> **Preferences.**

   If you are using the Wireless Toolkit with an IDE, see the documentation on the IDE you are using for information on how to access the Preferences tool and turn on profiling.

2. **In the Preferences dialog box, click the Monitoring tab and check the box to enable profiling.**

3. **Click OK in the Preferences window.**

4. **Run your application and then quit.**

   When you quit the application, the Profiler window opens displaying information collected during execution.

   Note – The profiling values obtained from an emulation does not reflect actual values on a real device, even though a real device skin might be used (such as, the `Motorola_i85s`).

Saving Profiling Information

To save profiling information:

1. **In the Profiler window, choose File** -> **Save or click Save in the toolbar.**

   You can also choose File -> Save As to rename a saved file.

2. **Type the name of the file in which you want to store the profiling information in the file chooser.**

Examining Saved Information

To examine previously saved profiling data from the Profiler window:

● **Click Open in the Profiler window's toolbar and select the file you want.**

To examine previously saved profiling data from the KToolBar:

1. **Choose File** -> **Utilities and click Open Session under Profiler.**

   If you are using the Wireless Toolkit with an IDE, see the documentation on the IDE you are using for information on how to access the Utilities tool.

2. **Select a previously saved file of profiling data from the file chooser.**

   The Profiler window opens displaying data from a previous data collecting session.

   To examine data for a specific method:

● **Click Find below the Call Graph and enter a string for the specific method you want to examine. Clicking Match Case does a search for the string as typed. Clicking Wrap returns to the beginning of the method hierarchy to continue the search.**

Examining Memory Usage

Another area to check for optimization is memory usage. The Memory Monitor Extension feature enables you to see how much memory is used by your application during runtime and to see a breakdown of the amount of memory usage per object.
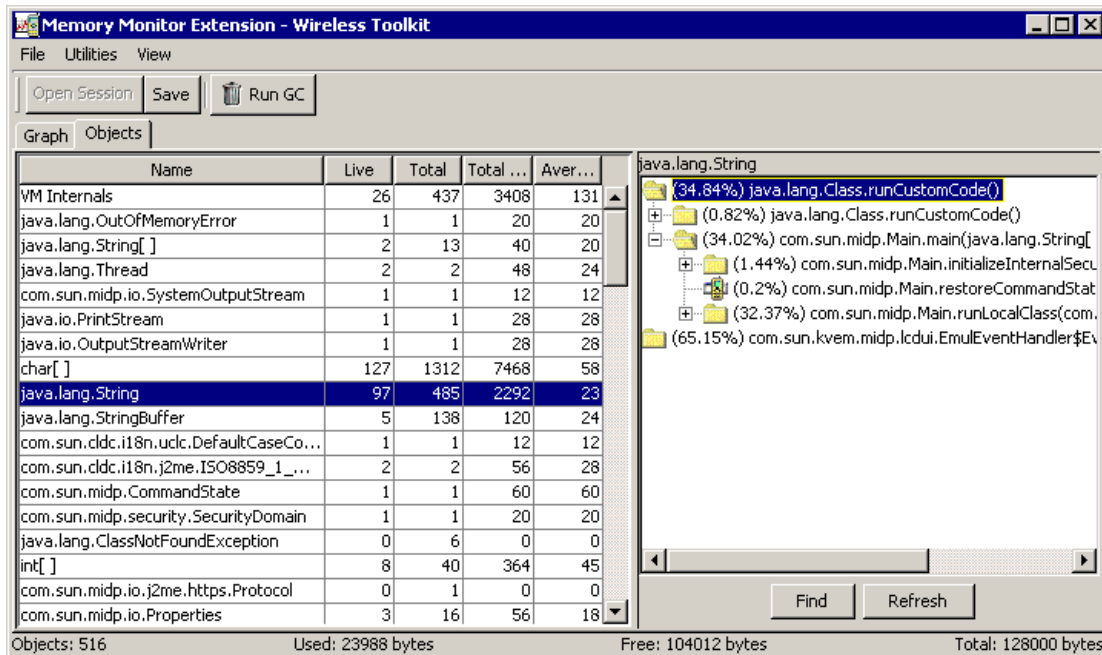
FIGURE 14   Memory Monitor Window

Memory Monitor Data Display

The Memory Monitor displays usage information in two tabbed panes:

- **Graph.** The Memory Usage graph displays:
  - Current. The current amount of memory used by the application.
  - Maximum. The maximum amount of memory used since program execution began. Denoted in the graph by a broken red line.
  - Objects. The number of objects in the heap.
  - Used. The amount of memory used.
  - Free. The amount of unused memory available.
  - Total. The total amount of memory available at startup (the sum of Used and Free).
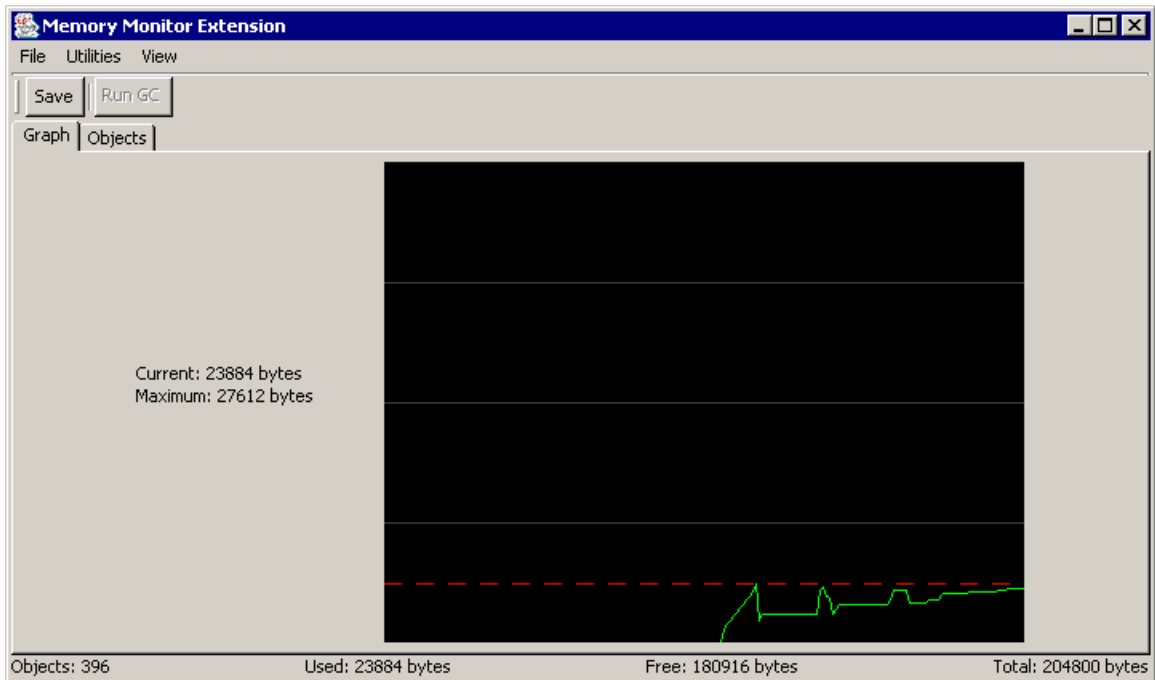
FIGURE 15   Memory Monitor Graph

- **Objects.** The Object Monitor breaks down the information into a table format that shows you:
  - Name. The name of each class examined for memory usage.
  - Live. The number of instances of an object in the heap.
  - Total. The total number of class objects allocated at startup.
  - Total Size. The total amount of memory used by the class' live objects.

■ Average Size. The average amount of memory used by a class live object with respect to the total size.
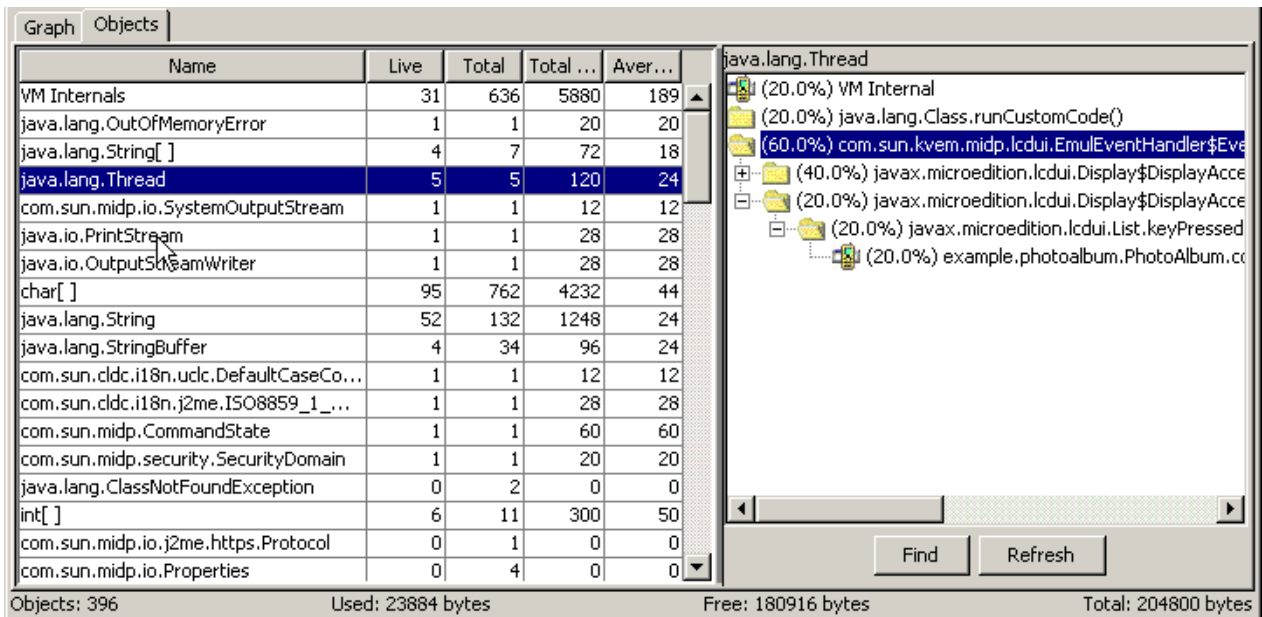
FIGURE 16   Memory Monitor Objects Table

You can click on the column titles in the table to sort the display. Clicking on Name sorts the classes in alphabetical order. Clicking Live, Total, Total Size, or Average Size sorts the information in that column from ascending to descending value.

Selecting a class in the Name column displays a hierarchical list of that class' methods and the percentage of memory used by the objects allocated by that method and the methods called by it in the pane to the right of the table. You can click Find to locate a specific method. The Objects table is dynamically updated during program execution; however, the method list is not. Click Refresh to update the display of percentage of usage information.

Viewing Memory Usage

To obtain memory usage information, follow these steps:

1. **From the KToolBar, choose Edit -> Preferences.**

   If you are using the Wireless Toolkit with an IDE, see the documentation on the IDE you are using for information on how to access the Preferences tool and turn on memory monitoring.

2. **In the Preferences dialog box, click the Monitoring tab.**

3. **Check the Enable Memory Monitor checkbox to turn on memory monitoring.**

   (Optional) If you want to set the heap size, click the Storage tab in the Preferences dialog box and enter a value. Acceptable values are between 32Kb and 64 Mbytes. Setting the heap size is not required to use memory monitoring.

4. **(Optional) Check the Excessive GC mode to turn on garbage collection before any memory is allocated.**

   This mode keeps memory usage to a minimum. When turned on, garbage collection is run every time an object is about to be allocated. Use this feature to determine the required amount of memory at any time during runtime.

5. **Click OK in the Preferences dialog box.**

6. **Run your application.**

   When execution begins, the Memory Monitor Extension window opens displaying memory usage as your application runs. You can request to have garbage collection performed at anytime while the application is running by clicking Run GC in the Memory Monitor Extension window.

   Note – The memory usage values obtained from an emulation does not reflect actual memory usage on a real device, even though a real device skin might be used (such as, the `Motorola_i85s`). The Memory Monitor merely provides you with possible indicators of excessive memory use for the emulation.

Saving Memory Usage Information

To save the information:

1. **In the Memory Monitor window, choose File -> Save or click Save.**

   You can also choose File -> Save As to rename a saved file.

2. **Type the name of the file in which you want to store the memory usage information in the file chooser.**

Examining Saved Information

To examine previously saved information from the Memory Monitor window:

● **Click Open Session in the Memory Monitor window's toolbar and select the file you want.**

To examine previously saved information from the KToolBar:

1. **From the KToolBar, choose File -> Utilities and click Open Session for Memory in the Utilities dialog box.**

   If you are using the Wireless Toolkit with an IDE, see the documentation on the IDE you are using for information on how to access the Utilities tool.

2. **Select a previously saved file of profiling data from the file chooser.**

Monitoring Network Traffic

One of the many uses of a MIDP application is to get or send information through the internet. By monitoring the network traffic generated by your application, you can obtain information you might need to improve or fix communication with a server and optimize network usage. The Network Monitor provides you with the following features:

■ Support for operating with or without an HTTP or HTTPS proxy
■ Viewing message information
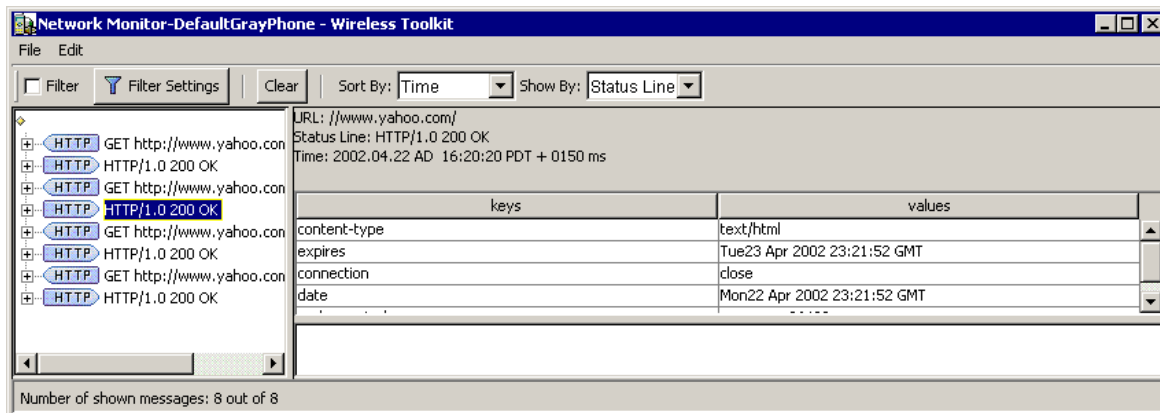
FIGURE 17   Network Monitor Window

Network Monitor Data Display

The Network Monitor displays a list of messages that were sent or received by the application. Messages are broken down into their main body and subparts, if any. You can examine the following message information:

- Selecting a message, displays its URL, status information (the protocol type and version), and the date and time the message was sent or received. The properties of the message are displayed in key and value pairs in a table format.

  You can see the entire contents of a value by choosing that value and viewing it in the scrollable text field at the bottom of the pane.
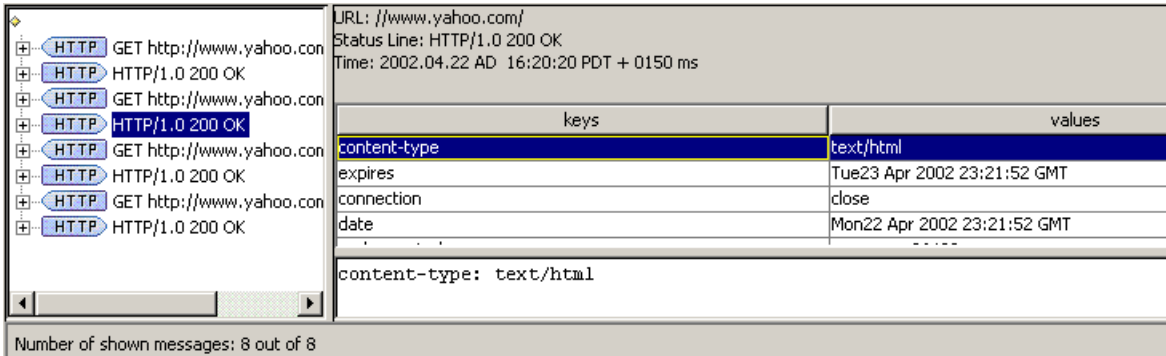
FIGURE 18 Message Key and Value Pair

- Selecting message body displays the hexadecimal values and the text value for the entire message.

  Nonprintable byte code is denoted by a "." in the text pane.

- Selecting a message chunk displays the hexadecimal value and text value for just that portion of the message.

  Nonprintable byte code is denoted by a "." in the text pane.
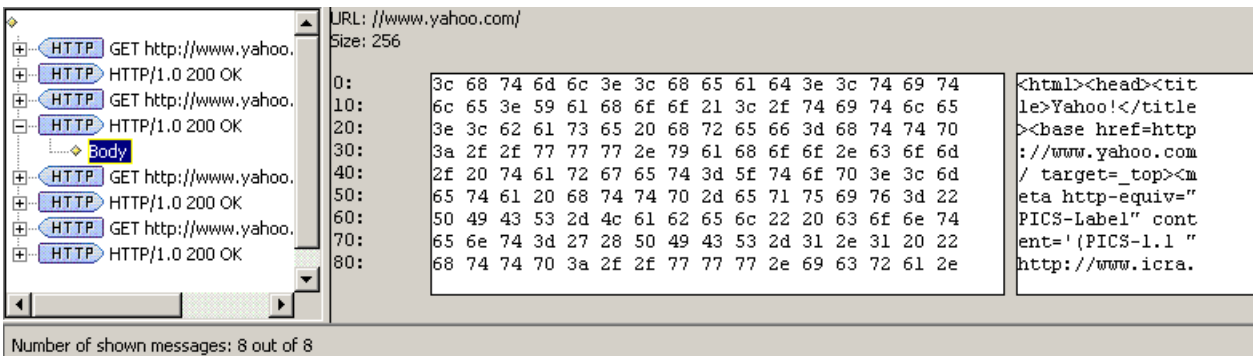
FIGURE 19 Message Body

> **Note –** You can examine messages that are still in the process of being sent. Incomplete messages are indicated by bold highlighting in the message tree.

Viewing Network Traffic

To turn on network monitoring automatically when your application runs or to choose a specific proxy type, follow these steps:

1. **From the KToolBar, choose Edit -> Preferences.**

   If you are using the Wireless Toolkit with an IDE, see the documentation on the IDE you are using for information on how to access the Preferences tool and turn on network monitoring.

2. **In the Preferences dialog box, click the Monitoring tab.**

3. **Check one of the protocol types, Enable HTTP Monitoring or Enable HTTPS Monitoring, to enable monitoring.**

4. **Click OK in the Preferences dialog box.**

   The Network Monitor window opens when you start your application.

Saving Message Information

To save the information collected by the Network Monitor:

1. **In the Network Monitor window, choose File -> Save or Save As.**

   You can also choose File -> Save As to rename a saved file.

2. **Type the name of the file in which you want to store the memory usage information in the file chooser.**

Examining Saved Messages

To examine previously saved message data from the Network Monitor window:

● **Choose File -> Open and select the file you want from the file chooser.**

   To examine previously saved message information from the KToolBar:

1. **From the KToolBar, choose File -> Utilities and click the Network Monitor button in the Utilities dialog box.**

   The Network Monitor window opens.

   If you are using the Wireless Toolkit with an IDE, see the documentation on the IDE you are using for information on how to access the Utilities tool.

2. **Choose File** -> **Open and choose the message file to examine from the file chooser.**

Saving a Networking Session

To save a networking session:

● **In the Network Monitor, choose File** -> **Save or Save As and type a file name in which to save your messages in the Save dialog box.**

Clearing the Message Tree

To remove the list of message in the message tree:

● **Choose Edit** -> **Clear or click Clear in the Network Monitor toolbar.**

Filtering Messages

To examine a specific set of messages, you can set filters in the Network Monitor. Only those messages that fall within the filter settings are displayed in the message tree.

1. **Choose Edit** -> **Filter Settings or click the Filter Settings button in the toolbar.**

2. **Change one or all of the filter settings in the Message Filter dialog box:**
   ■ Select the protocol type to examine, All, HTTP, or HTTPS in the Protocol text field.
   ■ Type the URL for the messages you want to see in the URL text field.
   ■ Type the status type of message you want to examine, in the Status Line text field.
   ■ Type the specific header in the Header Text text field.
   ■ Type a character string for the specific text in the body of the messages you want to examine in the Body Text text field.

Disabling Filtering

To disable message filtering so that all messages are displayed:

● **Click the Filter checkbox in the Network Monitor's button bar.**

Sorting Messages

To arrange the messages in the message tree in a particular order:

● **Open the Sort By combo box (click the Down arrow) and select one of the sort criteria:**

   ■ **Time**. Messages are sorted in chronological order of time sent or received.

   ■ **URL**. Messages are sorted by URL address. Multiple messages with the same address are sorted by time.

   ■ **Connection**. Messages are sorted by communication connection. Messages using the same connection are sorted by time. This sort type enables you to see messages grouped by requests and their associated responses.

Viewing Messages

To view a message by either its URL or status line:

● **Select URL or Status Line from the Show By combo box.**

   Messages in the tree are displayed by either their URL site or their status information.

Managing Device Speed

If the application you develop has a graphical user interface(GUI), the time required to draw the GUI on the screen is critical to the overall usability of the application. Another critical time factor is knowing the speed at which your application runs on a device. The VM emulation speed approximates the slower running speed of an application on a device. How quickly an application is able to transmit information to the network is another performance factor The Wireless Toolkit lets you modify both graphic speed emulation, VM speed emulation, and the speed of the network throughput.

The intent of the speed emulation features is to enable you to scale down the performance of some of the emulator subsystems to better reflect performance on a real device. Developing the application in a slower performance environment enables you to monitor and optimize the code of low-end devices. It is not the purpose of the speed emulation features to accurately emulate a specific device.
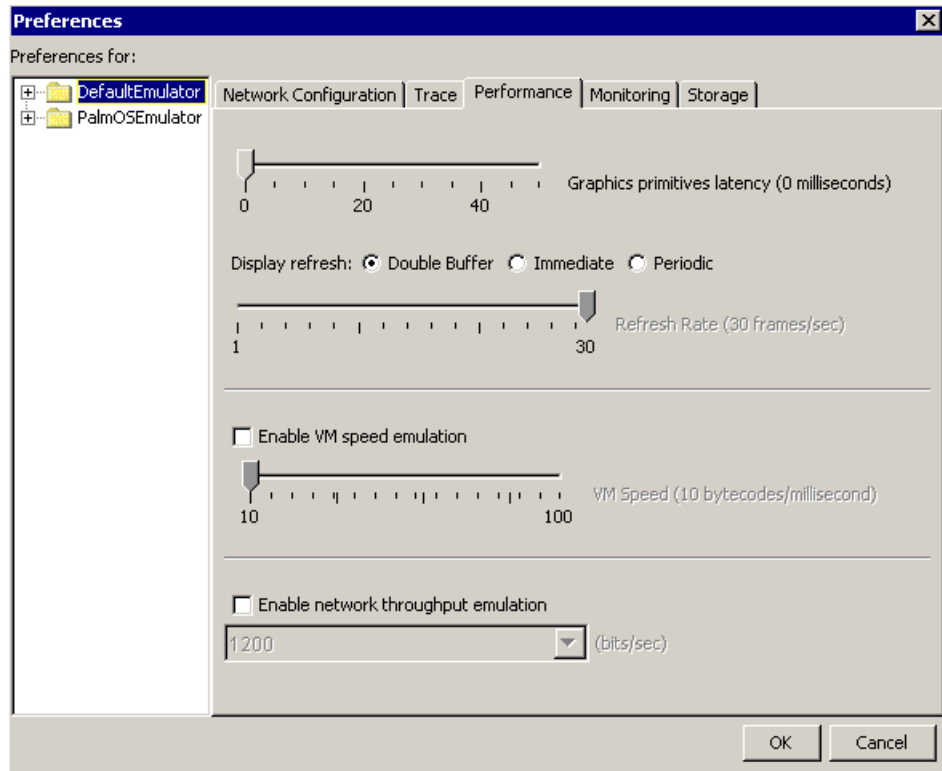
FIGURE 20   Performance Settings

Setting Performance Parameters

To adjust drawing and refresh speeds:

1. **Choose Edit -> Preferences, click the Performance tab and adjust the graphic rendering and repaint rates.**

   In the Performance tabbed pane, you can change one or more performance parameters. To optimize GUI display capabilities, you should adjust both the Graphics primitive latency speed and the Refresh mode:

   - **Graphics primitive latency**. The span of time in milliseconds for a graphic element to appear once the request is sent.
   - **Display refresh mode**. The number of times per second a device's screen is updated. There are three refresh modes: Double Buffer, Immediate, and Periodic.

   Double Buffer mode implements double buffering where a graphic is first rendered to an offscreen buffer and then copied to the screen. Immediate mode renders the graphic directly to the screen. Periodic mode lets you set the frequency in frames per second that the screen is refreshed.

2. **Click Ok in the Preferences dialog box and run your application.**

   Vary the latency value and the Refresh mode to find the settings that produce the fastest rendering with the least amount of flickering in your application.

   If you are using the Wireless Toolkit with an IDE, see the documentation on the IDE you are using for information on how to access the Preferences tool and set the graphic performance parameters.

Setting VM Speed Parameters

When running an emulation of a MIDlet, you cannot get an accurate demonstration of real time application execution speed. The emulation runs much faster than an application on an actual device. You can, however, adjust the VM speed emulation in the Wireless Toolkit to approximate the slower speed of a device on which the application might run.

Note – Setting the VM speed parameters does not emulate real device speed, even though a real device skin might be used (such as, the `Motorola_i85s`).

To set the VM speed emulation, which is the amount of Java byte code that is executed per second:

1. **Choose Edit -> Preferences and click the Performance tab.**

   If you are using the Wireless Toolkit with an IDE, see the documentation on the IDE you are using for information on how to access the Preferences tool and set the VM speed parameters.

2. **Click the Enable VM speed emulation checkbox and move the slider to the desired rate of speed.**

3. **Click OK in the Preferences dialog box and run your application.**

Setting Network Speed Parameters

Sometimes an application's performance is hindered by the speed of the network. To see how your application performs on a slow network, you can vary the network speed parameter. To set the rate at which the application transmits information to the network:

1. **Choose Edit -> Preferences and click the Performance tab.**

   If you are using the Wireless Toolkit with an IDE, see the documentation on the IDE you are using for information on how to access the Preferences tool and set the network speed parameters.

2. **Click the Enable network throughput emulation checkbox and select the desired rate of speed from the combo box.**

3. **Click OK in the Preferences dialog box and run your application.**

Note – Setting the network throughput speed does not emulate actual network transmission speed.

Working With the Emulator

The Emulator shows, on your computer, how your MIDP applications operate on a variety of mobile devices. Consequently, you can test your applications using the same platform you use to develop them, and defer testing on real devices until later in the development process.

The Emulator supports testing with several key features:

- **A variety of devices.** The Emulator can simulate several devices that take on a variety of form factors: cell phones, pagers, and palmtops. This helps you learn what type of experience users can expect and verify the portability of your application across different devices.

- **Tracing and debugging capabilities.** The Emulator supports run-time logging of various events, such as garbage collection, class loading, method calls, and exceptions. You can also perform source-level debugging with an IDE while an application runs in the Emulator.

- **Performance tuning**. The Emulator enables you to collect information that you can examine to optimize the performance of your MIDP applications. Performance tuning utilities include profiling methods and monitoring memory usage and network traffic. Also included is the ability to adjust the speed settings for graphic rendering and refreshing as well as VM speed emulation and network throughput. See the chapter, Chapter 4, "Performance Tuning Applications" for details on using the performance tools.

It must be emphasized that having an Emulator does not completely free you from testing on your target devices. The Emulator does have some limitations:

- **Accuracy of emulation.** The Emulator can only approximate a device's user interface, functionality, and performance. For example, the Emulator cannot simulate processing speed, so an application may run faster or slower on a target device than it does on the Emulator.

- **Application management.** The Emulator includes a sample implementation of an application manager, which supports the installation, maintenance, and removal of applications on a device. However, this sample implementation does not emulate any application manager in particular, as the behavior of application managers varies from device to device.

Example Devices

The J2ME Wireless Toolkit includes emulations of various example devices to help you test the portability of your application. The examples possess a range of displays and features found in mobile information devices, and they all support the MIDP specification.

TABLE 3   Example Devices

Tag	Description
DefaultColorPhone	Generic telephone with a color display.
DefaultGrayPhone	Generic telephone with a gray-scale display.
MinimumPhone	Generic telephone with minimum display capabilities.
RIMJavaHandheld	RIM device from Research In Motion Ltd.
Motorola_i85s	Motorola i85s telephone from Motorola, Inc.
PalmOS_Device	Palm OS personal digital assistant from Palm, Inc. (The emulation uses the Palm OS Emulator from Palm, Inc.)

This section describes the devices in more detail, and how to input text on these devices. For information about how to add more device definitions to the Emulator, see the *J2ME Wireless Toolkit Basic Customization Guide*, which comes with the J2ME Wireless Toolkit.

Device Characteristics

Some of the emulated devices, such as DefaultColorPhone, are generic examples of devices, whereas others, such as Motorola_i85s, approximate real devices that implement MIDP. The following table shows in more detail how the emulated devices differ.

TABLE 4   Selected Device Characteristics

Device Tag	Display Resolution	Color Support	Input Mechanism(s)	Number of Soft Buttons	Special Keys
DefaultColorPhone	96x128	256 colors	ITU-T keypad	2	
DefaultGrayPhone	96x128	256 shades of gray	ITU-T keypad	2	
MinimumPhone	96x54	Black & white	ITU-T keypad	0	BACK, MENU

TABLE 4   Selected Device Characteristics

Device Tag	Display Resolution	Color Support	Input Mechanism(s)	Number of Soft Buttons	Special Keys
Motorola_i85s	111x100	Black & white	ITU-T keypad	2	MENU
RIMJavaHandheld	198x202	Black & white	QWERTY keyboard	0	MENU
PalmOS_device	Variable, usually 160x160	Variable: Black & white up to 16-bit color	Graffiti and hard buttons	0	MENU, HOME

The emulated devices also differ in the ways they map game actions (UP, DOWN, LEFT, RIGHT, FIRE, A, B, C and D) and abstract commands.

In general, the MIDP implementation provided with the J2ME Wireless Toolkit works on all the example devices. However, some MIDP UI components, such as `javax.microedition.lcdui.DateField`, are unusable on devices with smaller displays than those of the `DefaultColorPhone` and `DefaultGrayPhone`.

The following sections describe in more detail how each of the devices work.

`DefaultColorPhone` and `DefaultGrayPhone`

The `DefaultColorPhone` device is a generic device representing a MIDP-enabled cellular phone with a color screen. The `DefaultGrayPhone` is identical in all aspects except for its screen, which is grayscale.

FIGURE 21 Default Color Phone Device

The interface for both devices includes:

- Buttons for the digits from 0 to 9, as well as pound and asterisk keys.
- Two soft buttons.
- A directional keypad, including a SELECT button in the center.
- SEND, END and CLEAR buttons.

Command menus are displayed by clicking the soft button Menu when it is displayed. The same soft button is used to hide the menu. The other soft button can still be used while the menu is displayed. The keys 7, 9, pound and asterisk are used for the game actions A, B, C and D. SELECT is used for the game action FIRE.

MinimumPhone

The `MinimumPhone` device represents the least capable device on which a MIDP application can be expected to run. Use this device when testing the adaptability of your application to small displays, and be prepared for the possibility that your application may not work as well on this device.

FIGURE 22 Minimum Phone Device

The device interface includes:

- Buttons for the digits from 0 to 9, as well as pound and asterisk keys.
- Four directional arrow keys (UP and DOWN are represented as being on the top and the bottom of a roller between the left and right arrow keys).
- A SELECT button (the center of the roller).
- A BACK button, used for MIDP abstract commands of type BACK
- A MENU button, used to show and hide the command menu.
- SEND and END buttons.

Command menus are displayed and hidden with the MENU key. The BACK key retains the same meaning when the command menu is displayed as when it is not. The keys 7, 9, pound and asterisk are used for the game actions A, B, C and D. SELECT is used for the game action FIRE.

Motorola_i85s

The `Motorola_i85s` device emulates many but not all aspects of the i85s telephone from Motorola, Inc. One aspect that the emulation does not represent accurately is the look-and-feel of the screen UI; what you see is based on the MIDP reference implementation and not on the Motorola MIDP implementation.

FIGURE 23 Motorola i85s Device

The device interface includes:

- Buttons for the digits from 0 to 9, as well as pound and asterisk keys.
- A directional keypad.
- Two soft buttons.
- A MENU button.
- A SEND button, that functions the same as the SELECT button on the devices described above.

The END button is not used on the `Motorola_i85s` device.

The behavior of the MIDP command menu is slightly different for Motorola_i85s than for the other devices. When the MENU button is pressed, the soft button labels are changed to BACK (for leaving the menu) and SELECT (for choosing a menu item). The keys 7, 9, pound and asterisk are used for the game actions A, B, C and D. SELECT is used for the game action FIRE.

RIMJavaHandheld

The `RIMJavaHandheld` device emulates many but not all aspects of the RIM Java Handheld device from Research In Motion, Limited. One aspect that the emulation does not represent accurately is the look-and-feel of the screen UI; what you see is based on the MIDP reference implementation and not on the RIM MIDP implementation.

FIGURE 24   RIM Java Handheld Device

This device has the following keys:

- Keys for the letters from A to Z in a QWERTY keyboard, the digits from 0 to 9, and various other symbols.
- UP and DOWN directional keys, on the top and bottom of the roller on the top right of the device.
- A MENU key, in the center of the roller.

- A BACK key, below the roller on the right hand side of the device
- An ENTER key.
- A SPACE key that also functions as a SELECT key.
- A BACKSPACE key.
- CAPS and ALT keys for changing the effect of key presses when entering text.

The NUM and DEL keys are not used in this emulation.

The keys V and B are used as left and right directional arrows. The keys Q, W, E and R are used for the game actions A, B, C and D. SPACE is used for the game action FIRE.

PalmOS_Device

The `PalmOS_Device` emulation uses the Palm OS Emulator (POSE) from Palm, Inc. to emulate devices running Palm OS. (For instructions on obtaining POSE and configuring it to work with the J2ME Wireless Toolkit, see Chapter 2, "Installing the Wireless Toolkit.")
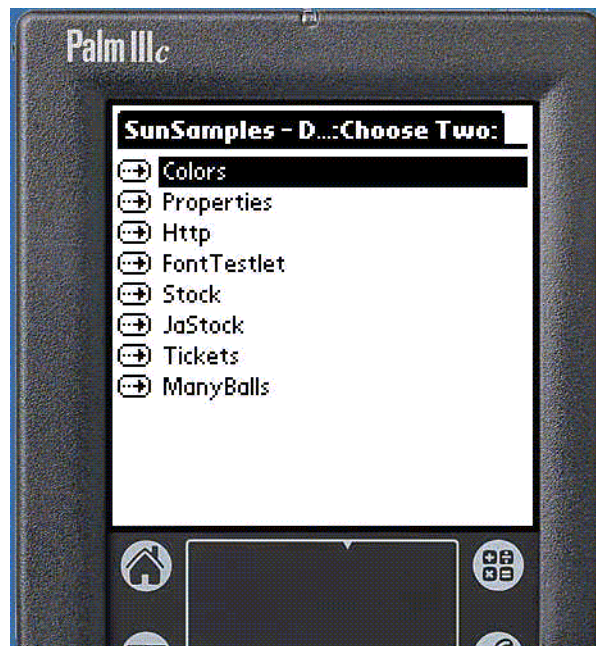

FIGURE 25   Palm OS Device

The `PalmOS_Device` emulation has a user interface substantially different from that of the other devices:

- The Graffiti handwriting recognition system and the native Palm keyboard are used for text input.

- The screen is touch sensitive; it reacts to mouse clicks.
- The up and down arrow keys, together with the buttons to their left and right, are used for navigation.
- The leftmost and rightmost buttons on the device both function as SELECT and as the game action FIRE.
- Command menus are activated by clicking (tapping) the menu icon on the lower left of the device's screen.
- The four game actions are performed by clicking on each of the four corners of the Graffiti text input area.

To configure the user interface, use the J2ME Wireless Toolkit Preferences tool. (See "PalmOSEmulator Preferences" on page 56.) Alternatively, you can select Java Preferences from the Options menu when running a MIDP application on POSE. (For more information, consult the MIDP for Palm OS documentation, available through `http://java.sun.com/products/midp/palmOS.html`.)

Inputting Text

When a MIDP application needs character input from the user, it displays a text box. For each of the bundled devices, with the exception of `PalmOS_Device`, you can enter text into this box using the buttons in the interface of the device or the keyboard of your computer.

See Appendix C, "Internationalization" for information about using different fonts with the Emulator.

Using the Device to Input Text

You can use the keypad of the `DefaultColorPhone`, `DefaultGrayPhone`, `MinimumPhone`, `Motorola_i85s`, and `RIMJavaHandheld` devices to input text. When you begin entering text in this manner, the device screen displays a text box.

The functions of the pound ('#') and asterisk ('*') keys vary depending on the type of input being requested:

TABLE 5   Pound ('#') and Asterisk ('*') Key Functions

Input type	Pound key function	Asterisk key function
Phone number	Pound ('#')	Asterisk ('*')
Numeric	Minus sign ('-')	None
All other types	Switches input mode between upper case, lower case, numeric and symbol mode.	Space

See the API documentation for `javax.microedition.lcdui.TextField` for details on MIDP input constraints.

Using the Keyboard to Input Text

You can also enter text in a MIDP application using the keyboard of your computer. When you begin entering text from the keyboard, the image of the device's screen is replaced by an area in which you can enter text.

When you have finished entering text, press Enter on the keyboard. The Emulator returns to the J2ME application.

Note – You can also switch to and from keyboard input mode using the device's SELECT key.

Application Demos

The toolkit comes with the following demo applications, which can all be run in the Emulator:

- demos, a set of seven MIDlets:
    - Colors, a small MIDlet demonstrating the use of colors.
    - Properties, a MIDlet providing information about the VM and device.
    - Http, a small MIDlet demonstrating the use of HTTP connections.
    - FontTestlet, a small MIDlet demonstrating the use of fonts.
    - Stock, a client-server stock ticker.
    - Tickets, a concert ticket bidding system.
    - ManyBalls, a small MIDlet demonstrating UI and threads with bouncing balls.
- games, a set of three MIDlets:

- ■ TilePuzzle, a word shuffling game.
  - ■ WormGame, a graphical action game.
  - ■ PushPuzzle, a puzzle game.
- ■ PhotoAlbum, a single MIDlet demonstrating graphics support.
- ■ UIDemo, a single MIDlet demonstrating MIDP high-level UI widgets.

The demo applications are all written using the MIDP APIs but they are not all portable across all the devices. They have been optimized to run on the `DefaultColorPhone` and the `DefaultGrayPhone` device types. Their appearance degrades when run on the `MinimumPhone` device type. For example, the PushPuzzle game is portable, but the TilePuzzle game is not, and the Stock and Tickets applications primarily make use of high-level GUI components.

For information on how to demonstrate your MIDlet applications for non-development purposes, see Appendix B, "MIDlet Demonstration."

Selecting a Default Device

If you do not specify which device to emulate, the Emulator uses the default device, `DefaultGrayPhone`, when you run a MIDlet. To change the default emulated device:

1. **From the Windows Start menu, select Programs -> J2ME Wireless Toolkit 1.0.4 -> Default Device Selection.**

   The Default Device Selection dialog appears with a menu of devices.

2. **Select the device from the menu, and press OK.**

   The next time you run a MIDlet, it will be emulated on the device you have chosen.

Preferences and Utilities

This section describes the Preferences and Utilities tools, which are used to configure the device emulators and run utilities specific to those emulators.

You can use the Preferences and Utilities tools from within a development environment, such as KToolBar. You can access the Utilities tools from the KToolBar's File menu. You can access the Preferences tools from the KToolBar's Edit menu.

Alternatively, you can use the tools from the Microsoft Windows Start menu. To start the Preferences tool from the Microsoft Windows Start menu:

- ● **Choose Programs -> J2ME Wireless Toolkit 1.0.4 -> Preferences.**

To run the Utilities tool:

● **Choose Programs** -> **J2ME Wireless Toolkit 1.0.4** -> **Utilities.**

If you are using the Wireless Toolkit with an IDE, see the documentation on the IDE you are using for information on how to access the Preferences and Utilities tools.

Device Categories

The Preferences and Utilities tools divide the emulated devices into two categories:

■ `DefaultEmulator`. This category includes the devices `DefaultColorPhone`, `DefaultGrayPhone`, `MinimumPhone`, `RIMJavaHandheld` and `Motorola_i85s`.

■ `PalmOSEmulator`. This category has only one device, the `PalmOS_Device`.

The following sections describe how the available preferences and utilities for these categories.

DefaultEmulator Preferences

Use the DefaultEmulator Preferences dialog box to configure the `DefaultEmulator` devices.
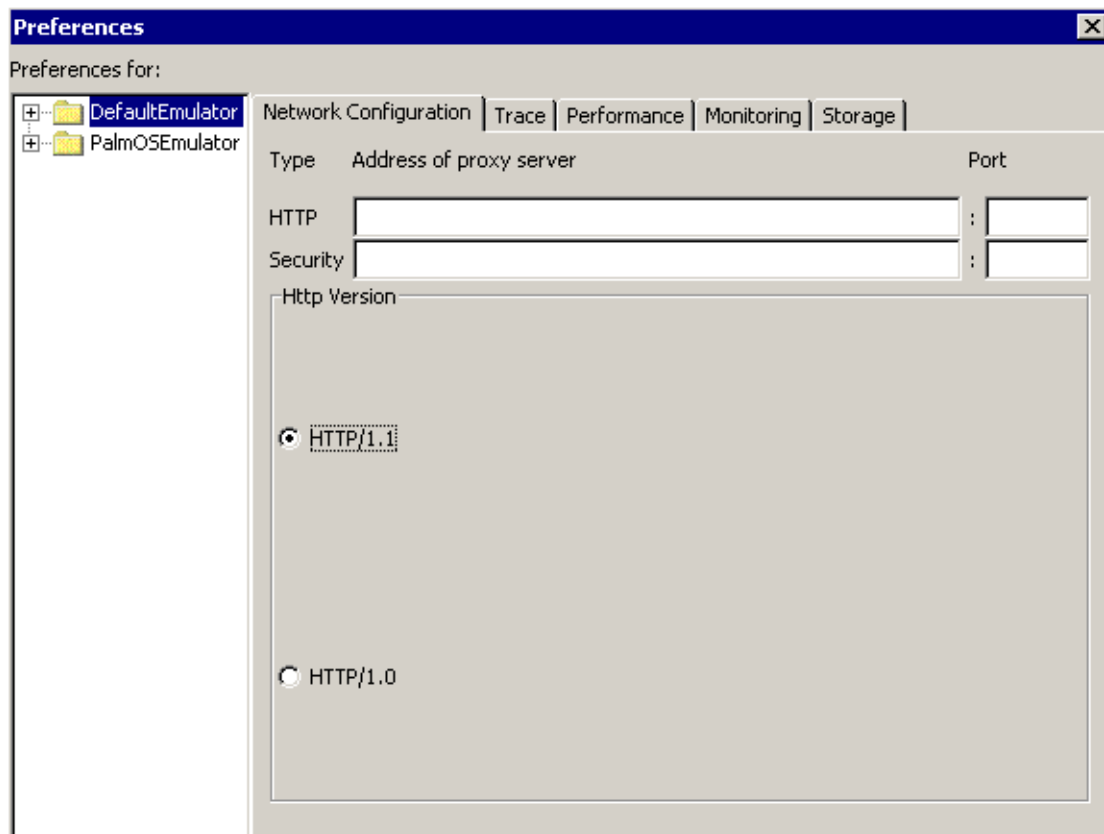
FIGURE 26   DefaultEmulator Preferences Dialog

Setting the Web Proxy

If you want to run Java applications that require Web connections, and if such connections can be made only through a proxy server (when the Web server is on the other side of a firewall, for example), then you need to configure the Emulator with proxy server information.

To specify proxy information for HTTP connections:

1. **Choose Edit -> Preferences and click the Network Configuration tab.**

2. **Type the name of the HTTP proxy server and its port number in the HTTP text fields.**

3. **Click Ok in the Preferences dialog box.**

To specify proxy information for HTTPS connections:

1. **Choose Edit** -> **Preferences and click the Network Configuration tab.**

2. **Type the name of the HTTPS server and its port number in the Security Server and Port text fields.**

3. **Click Ok in the Preferences dialog box.**

   If you are unsure about the correct proxy settings, ask your system administrator.

Choosing an HTTP Version

The Wireless Toolkit provides you with two versions of HTTP to work with. Version 1.0 is provided for development purposes only. The MIDP specification requires that HTTP version 1.1 is supported. Selecting version 1.0 in the Network Configuration tab of the Preferences dialog box disables some of the version 1.1 features, such as chunking messages and providing a persistent connection to enable you to work with servers that do not support version 1.1.

Setting the Heap Size

To specify the amount of heap memory to make available to MIDP applications:

1. **Choose Edit** -> **Preferences and click the Storage tab.**

2. **Type a value in the Heap Size field (in kilobytes).**
   The default value is 500 kilobytes.

Setting the RMS Directory

You can specify a Record Management System (RMS) directory prior to running your MIDlet. To set the directory:

1. **Choose Edit** -> **Preferences and click the Storage tab.**

2. **Type the name of the directory in which you want to store information the next time you activate a MIDlet.**
   The directory is created under the `appdb` directory. A database (.db) file is created and placed in the specified directory. The default directory location is the `appdb`. You must create a unique directory for each session you want to save.

Enabling Tracing

You can configure the Emulator to trace certain types of events:

- **Garbage collection.** The trace output indicates when garbage collection occurs, as well as the number of bytes collected and the total heap size.

- **Class loading.** The trace output displays the name of every non-system class as it is created and initialized.
- **Method calls.** The trace output has an entry for each method call, recording the name of the method and the object on which it was invoked. Note that this output is very verbose and may cause your application to run slowly.
- **Exceptions.** The trace output includes a record of every exception that is thrown, including those that are thrown and caught by system classes.

To enable (or disable) tracing of any of these events, check (or uncheck) the corresponding boxes in the Trace tab in the Preferences dialog box.

DefaultEmulator Utilities

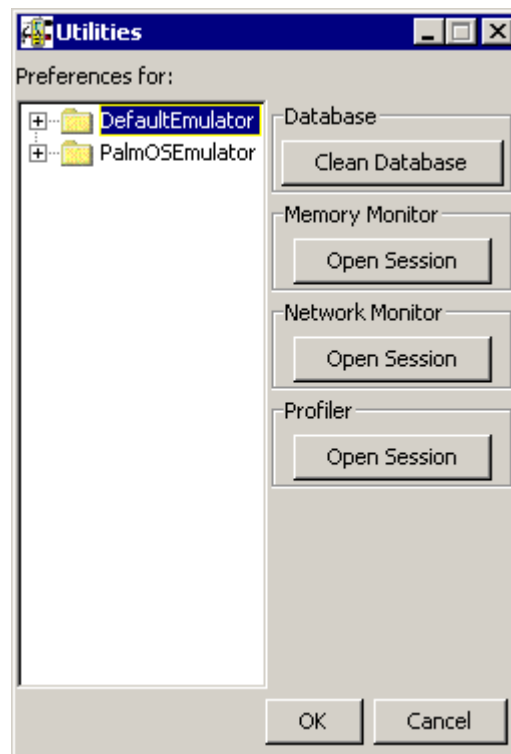You can use the DefaultEmulator Utilities window to run the `DefaultEmulator`'s utilities.

FIGURE 27 Default Emulator Utilities Window

Cleaning Device Storage

The `DefaultEmulator` simulates a client device's local storage by maintaining small database files on your computer. To erase these database files, click Clean Database.

Monitoring Memory Usage

You can see where a bottleneck in your application's performance might be occurring by reviewing its memory usage during runtime. The Memory Monitor provides several kinds of memory usage information, such as the amount of memory that live class objects use during program execution. For information on how to use the Memory Monitor, see "Examining Memory Usage" on page 28 in Chapter 4, "Performance Tuning Applications."

Profiling Methods

The Profiler collects data from the DefaultEmulator during runtime. By seeing how much time a method takes to execute, you can see what areas of your application could be slowing down execution time. For information on examining profiling information, see "Profiling Your Application" on page 25, in Chapter 4, "Performance Tuning Applications."

Monitoring Network Traffic

You can use the `DefaultEmulator` to simulate the transmission of messages to and from a device and the internet. For information on how to use the Memory Monitor, see "Monitoring Network Traffic" on page 32 in Chapter 4, "Performance Tuning Applications."

`PalmOSEmulator` Preferences

You can use the PalmOSEmulator Preferences to configure the `PalmOSEmulator` devices.
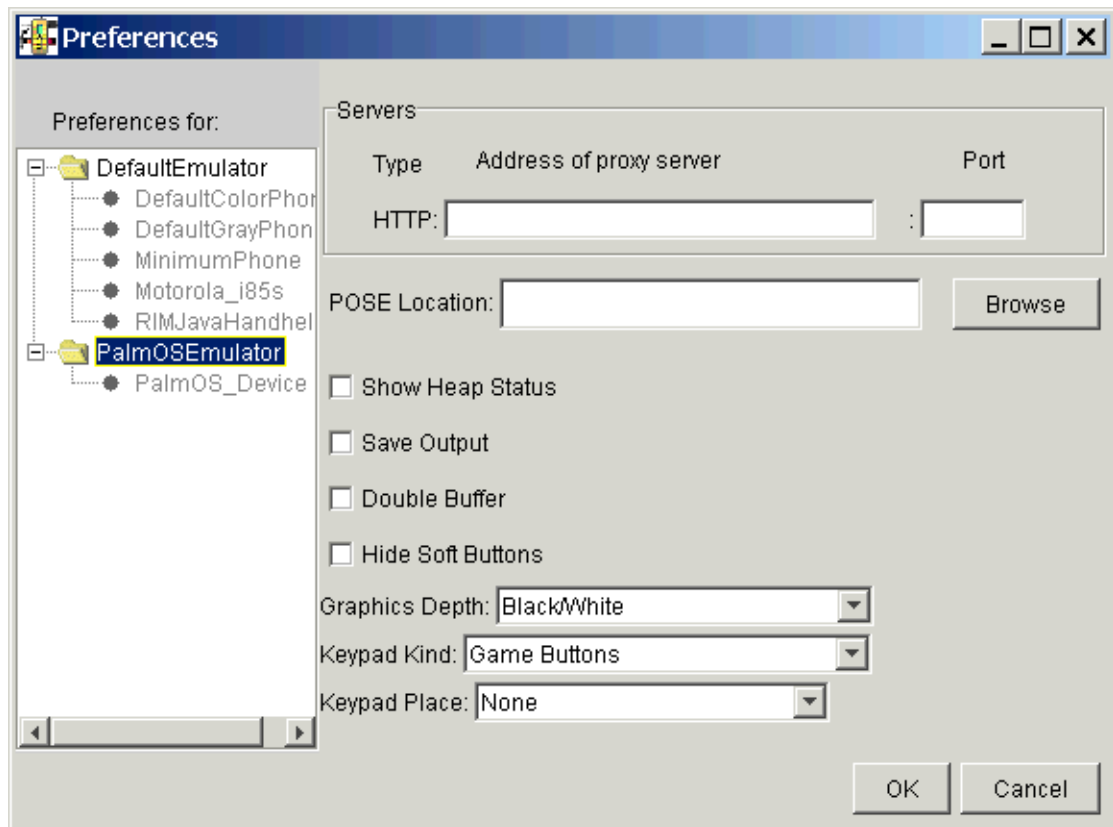
FIGURE 28   PalmOSEmulator Preferences Window

Setting the Web Proxy

Like the `DefaultEmulator`, the `PalmOSEmulator` lets you specify an HTTP proxy. (The `PalmOSEmulator` does not, however, use HTTPS proxies.)

To specify proxy information for HTTP connections, enter the name of the server and its port using the HTTP Address of Proxy Server and Port fields.

If you are unsure about the correct proxy settings, ask your system administrator.

Setting the POSE Location

The J2ME Wireless Toolkit needs the `PalmOSEmulator` (POSE) in order to emulate a MIDP application running on a Palm OS device. To specify where the POSE is installed on your system, press Browse beside the POSE Location text field and use the file chooser to find and select the POSE executable.

Showing the Heap Status

To set the Emulator to show the status of the heap memory before running a MIDlet, check the Show Heap Status box in the Preferences dialog.

Saving Application Output

To save the output of the Emulator for later viewing, check the Save Output box in the Preferences dialog. Any data written to the standard output stream and standard error stream appears in the files STDOUT.txt and STDERR.txt; the placement of these files depends on how you run POSE. For more information, consult the MIDP for Palm OS documentation, available through http://java.sun.com/products/midp/palmOS.html.

Enabling Double Buffering

For the PalmOS_Device, output from a MIDlet to the screen is not buffered, so the screen is updated every time the application draws a line or writes some text. To enable buffering, check the Double Buffer box in the Preferences dialog.

Hiding the Soft Buttons

Soft buttons are displayed above the writing area. To remove the buttons, so that there is more space for the application, check the Hide Soft Buttons box in the Preferences dialog. (You can use the application's menus to do anything you might do with a soft button.)

Setting the Graphics Depth

To set the number of colors used to display applications, select a value from the Graphics Depth combo box. The range of color settings spans from black and white to millions of colors.

Showing the Keypad

Use the Keypad Kind menu to specify which keys, if any, should be placed on the screen. Your options are:

■ Game Buttons. Hides the controls. You control the application using the buttons and graffiti area on your Palm OS device. This is the default setting.

■ Small. Displays just the arrow keys and SELECT/FIRE button along one side of the screen.

■ Full. Displays a phone pad, arrow keys, SELECT/FIRE button, and game actions (A, B, C, D) along one side of the screen.

If you select the Small or Full keypad, then use the Keypad Place menu to place the keypad on the left or right side of the screen.

For more information, see the *MIDP for Palm OS User's Guide.*

`PalmOSEmulator` Utilities

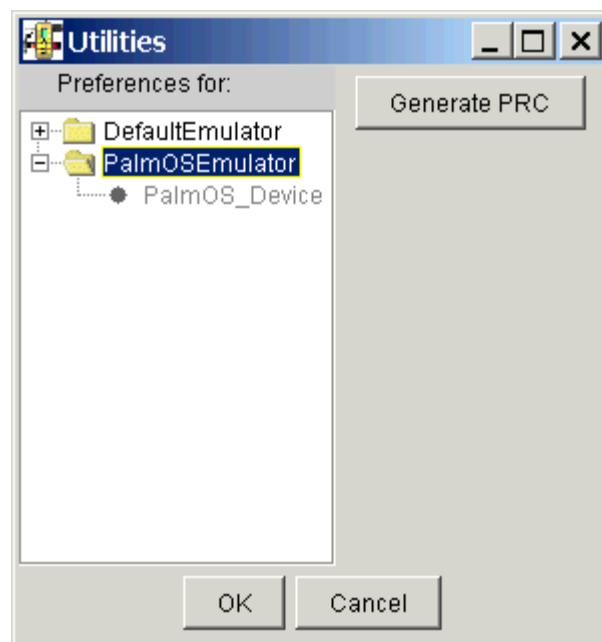You can use the PalmOSEmulator Utilities to generate Palmpilot resource files (PRC).

FIGURE 29 PalmOSEmulator Utilities Window

Generating PRC Files

PRC is the file format used to transfer a packaged application from a desktop computer to a Palm OS device. To generate a PRC file from your MIDP application:

● **Choose File** -> **Utilities, select PalmOSEmulator and click Generate PRC. The MIDP for Palm OS PRC Converter appears. (For more information, see the** *MIDP for Palm OS User's Guide.***)**

Operating From the Command Line

This chapter describes how to operate the J2ME Wireless Toolkit tools from the command line and details the steps required to build and run an application.

Preliminary Checks

Before building and running an application from the command line, type
`java -version` at the command line to verify that the J2SE `bin` directory, for example, `C:\jdk1.3\bin`, is in your `PATH`. The reply should show that the version of the J2SE SDK that you are using is version 1.3.0 or higher.

For more examples, see the files `build.bat` and `run.bat` in the `bin\` directories of the demonstration applications. You can find these files under the
{*j2mewtk.dir*}`\apps\`{*demo_name*}`\bin\` directory where {*j2mewtk.dir*} is the installation directory of the J2ME Wirless Toolkit and {*demo_name*} is the name of one of the demo applications.

Accessing Preferences and Utilities

To access the Preferences and Utilities tools discussed in and from the command line, type the following commands at the prompt:

{*j2mewtk.dir*}`\bin\prefs.exe`

{*j2mewtk.dir*}`\bin\utils.exe`

Compiling Class Files

J2ME class files are compiled from Java source files ising the javac compiler from the J2SE SDK. Before compiling, you should verify that the following subdirectories exist and create them if necessary:

1. `tmpclasses`. A directory to hold unverified classe.

2. `classes`. A directory to hold verified classes.

To compile an application, use the javac command as follows (all on one line):

`javac [`*options*`] -bootclasspath {`*j2mewtk.dir*`}\lib\midpapi.zip` *<files>*

Arguments

<files>

A list of one or more source files to compile, separated by spaces.

Options

-d *<output directory>*

Specify the directory into which the compiler should output classes. (This directory must exist before compiling.)

Note – If you are using the compiler included with the Java 2 SDK, Standard Edition 1.4, use the `-target` 1.1 option when compiling your source files. If you do not do this, you will have problems preverifying your compiled classes.

Example

To compile all the source files located in the `src` directory (but not its subdirectories) and place them into the directory `tmpclasses`, use the following command:

```
javac -d tmpclasses -bootclasspath c:\wtk104\lib\midpapi.zip
    -classpath tmpclasses;classes src\*.java
```

The *tmpclasses* directory is used to store the compiled classes while they are not yet verified. After verification has been performed, the preverifier stores the classes in the *classes* directory. For more information about the javac command, see the J2SE SDK documentation.

Preverifying Classes

To preverify application classes, use the preverify command that comes with the J2ME Wireless Toolkit. The synatx for the preverify command is as follows:

```
preverify [options] <files | directories>
```

Arguments

<files | directories>

A list of one or more files or directories to preverify, separated by spaces.

Options

`-classpath` *<classpath>*

Specify the directories or JAR files (given as a semicolon-delimited list) from which classes are loaded.

`-d` *<output directory>*

Specify the directory into which the preverifier should output classes. (This directory must exist before preverifying.) If this option is not used, the preverifier places the classes in a directory called output.

Example

Following the example in the previous section, after compiling the source files, use the following command:

```
preverify -classpath c:\wtk104\lib\midpapi.zip tmpclasses -d
    classes
```

As a result of this command, pre-verified versions of the class files are placed in the classes directory.

Packaging a MIDlet suite

To package a MIDlet suite, you must first create a manifest file, then create an application JAR file, and finally, an application JAD file.

Creating a Manifest File

Create a manifest file containing the appropriate attributes as specified in
Appendix A, "MIDlet Attributes."

You can use any plain text editor to create the manifest file. A manifest might have
the following contents, for example:

```
MIDlet-1: My MIDlet, MyMIDlet.png, MyMIDlet
MIDlet-Name: MyMIDlet
MIDlet-Vendor: My Organization
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
```

Creating an Application JAR File

Create a JAR file containing the manifest as well as the suite's class and resource
files.

- **To create the JAR file, use the JAR tool that comes with the J2SE SDK. The
  syntax is as follows:**

  jar cfm *<file>* *<manifest>* -C *<class_directory>* . -C *<resource_directory>* .

Arguments

<file>

The JAR file to create.

<manifest>

The manifest file for the MIDlets.

<class_directory>

The directory containing the application's classes.

<resource_directory>

The directory containing the application's resources.

Example

To create a JAR file named MyApp.jar whose classes are in the classes directory
and resources are in the res directory, use the following command:

jar cfm MyApp.jar MANIFEST.MF -C classes . -C res .

Creating an Application JAD File

Create a JAD file containing the appropriate attributes as specified in Appendix A, "MIDlet Attributes." You can use any plain text editor to create the JAD file. This file must have the extension `.jad`.

Note – You need to set the MIDlet-Jar-Size entry to the size of the JAR file created in the previous step.

Example

A JAD file might have the following contents, for example:

```
MIDlet-Name: MyMIDlet
MIDlet-Vendor: My Organization
MIDlet-Version: 1.0
MIDlet-Jar-URL: MyApp.jar
MIDlet-Jar-Size: 24601
```

Running the Emulator

You can run the Emulator from the command line using the emulator command. Your current directory should be the `bin\` subdirectory of the directory where you installed the Wireless Toolkit, for example, `C:\WTK104\bin`. The syntax for the emulator command is as follows (all on one line):

```
emulator [options]
```

General Options

`-help`

Display a list of valid options.

`-version`

Display version information about the emulator.

`-Xquery`

Print device information on the standard output stream and exit immediately. The information includes, but is not limited to, device name, device screen size, and other device capabilities.

Running Options

`-Xdevice:<`*device name*`>`

Run an application on the device specified by the given device name. (For a list of device names, see TABLE 3 on page 42 in Chapter 5, "Working With the Emulator."

`-Xdescriptor:<`*descriptor file*`>`

Run an application using the given descriptor file.

`-classpath <`*classpath*`>`

Specify the classpath for libraries required to run the application.

`-D<`*runtime property*`>`

Set the HTTP and HTTPS proxy servers. Valid properties include:

`com.sun.midp.io.http.proxy=<`*proxy host*`>:<`*proxy port*`>`

Tracing and Debugging Options

`-Xverbose:<`*trace options*`>`

Display trace output, as specified by a list of comma-separated options:

> `classes`
>
> Trace class loading.
>
> `gc`
>
> Trace garbage collection.
>
> `all`
>
> Use all tracing options.

`-Xdebug`

Enable runtime debugging. The `-Xrunjdwp` option must also be used.

`-Xrunjdwp:`*<debug settings>*

Start a JDWP debug session, as specified by a list of comma-separated debug settings. The `-Xdebug` option must also be used. Valid debug settings include:

`transport=`*<transport mechanism>*

The transport mechanism used to communicate with the debugger. The only transport mechanism supported is `dt_socket`.

`address=`*<host:port>*

The transport address for the debugger connection. You can omit providing a *host*. If *host* is omitted, *localhost* is assumed to be the host machine.

`server=`*y|n*

Start the debug agent as a server. The debugger must connect to the port specified. The possible values are *y* and *n*. Currently, only *y* is supported (the Emulator must act as a server).

Emulator Preferences Setting Option

`-Xprefs:`*<filename>*

Set the Emulator preferences to the values in the given property file. The *filename* you provide should be the full path name of a property file, which is used to override the values in the preferences window. The property file can contain the following properties:

TABLE 6   Emulator Preferences Properties List

Property Name	Property Description and Legal Values
`http.version`	Network Configuration > HTTP Version Value: HTTP/1.1 | HTTP/1.0
`kvem.excessivegc`	Monitoring > Excessive GC Mode Value: true | false
`kvem.memory.monitor.enable`	Monitoring > Enable memory monitor Value: true | false
`kvem.netmon.http.enable`	Monitoring > Enable HTTP monitoring Value: true | false
`kvem.netmon.https.enable`	Monitoring > Enable HTTPS monitoring Value: true | false
`kvem.profiler.enable`	Monitoring > Enable profiling Value: true | false
`netspeed.bitpersecond`	Performance > bits/sec combo box Value: integer

TABLE 6   Emulator Preferences Properties List

Property Name	Property Description and Legal Values
`netspeed.enableSpeedEmulation`	Performance > Enable network throughput emulation Value: true | false
`screen.graphicsLatency`	Performance > Graphics primitives latency Value: integer
`screen.refresh.mode`	Performance > Display refresh (radio button) Value: default | immediate | periodic
`screen.refresh.rate`	Performance > Display refresh (slider) Value: integer
`vmspeed.bytecodespermilli`	Performance > Enable VM speed emulation (check box) Value: integer
`vmspeed.enableEmulation`	Performance > Enable VM speed emulation (slider) Value: true | false
`storage.root`	Storage > Storage root directory Value: String (relative path to *appdb*)
`storage.size`	Storage > Storage size Value: integer

Java Application Manager (JAM) Options

`-Xjam[:`*`<argument>`*`]`

Run an application using the example Java Application Manager. When no argument is supplied, the graphical JAM is launched, as described in "Running a Remotely-Deployed Application Using the Java Application Manager (JAM)" on page 72 in Chapter 7, "Testing Application Provisioning." Possible values for *<argument>* include:

`install=`*`<url>`*

Install an application from a URL.

`run=`*`<name | index>`*

Run a previously installed MIDlet, as specified by its storage name or index. If no application is specified, the graphical JAM will be run.

`remove=`*`<name | index>`*

Remove a previously installed MIDlet, as specified by its storage name or index. If no application is specified, the graphical JAM will be run.

`transient=`*`<url>`*

Install an application from a URL, run it, and then remove it. (Consider this option a shortcut for launching the emulator three separate times to install, run and then finally remove the application.)

The URL must point to the application's descriptor file.

`list`

Present a detailed list of all applications installed.

`storageNames`

Present the names of all applications installed, in an easy-to-parse format.

Examples

Here are some examples of running the Emulator from the command line:

- **To run the application described by the file** `c:\wtk104\apps\example\bin\example.jad`**, use the following command:**

  ```
  emulator -Xdescriptor:C:\J2MEWTK\apps\example\bin\example.jad
  ```

- **To run the PhotoAlbum MIDlet, whose classes are in the directory** `classes/`**, using the** `DefaultColorPhone` **device, use the following command:**

  ```
  emulator -Xdevice:DefaultColorPhone -classpath classes
  example.photoalbum.PhotoAlbum
  ```

- **To run the PhotoAlbum MIDlet, whose classes are in the directory** `classes/`**, tracing garbage collection and class loading, use the following command:**

  ```
  emulator -Xverbose:gc,class -classpath classes
  example.photoalbum.PhotoAlbum
  ```

● **To run the application described by** `example.jad` **and wait for a connection from a remote debugger on port 5000, use the following command:**

```
emulator -Xdebug
-Xrunjdwp:transport=dt_socket,address=<localhost>:5000,server=y
-Xdescriptor:example.jad
```

Where *localhost* is the name of the machine on which the debugger is run or you can specify just the port number without providing a *<localhost>* name.

Testing Application Provisioning

The document, "Over The Air User Initiated Provisioning Recommended Practice for the Mobile Information Device Profile," describes how MIDlet suites can be deployed over-the-air (OTA), and the functions that a device should provide to support such deployments. You can obtain this document at http://java.sun.com/j2me/docs/.

The MIDP implementation of the J2ME Wireless Toolkit's Default Emulator emulates the device behavior during the provisioning process. You can use this functionality to test and demonstrate the full provisioning process of the MIDlet suites from the server to the device. This chapter explains the different steps that are required to perform this process.

Deploying Applications on a Web Server

To deploy a MIDP application remotely on a Web server:

1. **Change the JAD file's MIDlet-Jar-URL property to the URL of the JAR file.**

   This URL must be absolute. For example:

   ```
   MIDlet-Jar-URL: http://mumble.java.sun.com/midlets/example.jar
   ```

2. **Ensure that the Web server recognizes JAD and JAR files:**

   a. **For the JAD file type, set the file extension to** `.jad` **and the MIME type to** `text/vnd.sun.j2me.app-descriptor`.

   b. **For the JAR file type, set the file extension to** `.jar` **and the MIME type to** `application/java-archive`.

   The details of how to configure a Web server depend on the specific software used.

Running a Remotely-Deployed Application Using the Java Application Manager (JAM)

J2ME-enabled devices include a Java Application Manager (JAM) for downloading, installing, and configuring J2ME applications. The Emulator has an example JAM you can use to demonstrate how the user would obtain and manage your application. This example JAM supports network delivery of applications, according to the recommended practice for MIDP (see the document, "Over The Air User Initiated Provisioning Recommended Practice" at http://java.sun.com/j2me/docs/ for a discussion of recommended practices).

You can use the Java Application Manager in one of the following ways:

- Emulate the process using the JAM's graphical user interface
- Perform a single operation from the command line with the JAM option

  For information on performing single operations through the command line, see "Java Application Manager (JAM) Options" on page 69 in Chapter 6, "Operating From the Command Line."

The rest of this section explains how to use the graphical JAM on a device. To run the GUI JAM, open a command prompt, and follow these steps:

1. **Change the current directory to the** {*j2mewtk.dir*}\bin**.**

   For example, the sequence of commands might be:

   ```
   C:
   cd \WTK104\bin
   ```

2. **Enter the following command:**

   ```
   emulator -Xjam
   ```

   The Emulator appears. When you advance past the copyright screen, you see the JAM's main screen:

FIGURE 30 JAM Main Screen

3. Click the Install soft button.

The Emulator asks you where the application is located.

FIGURE 31 Text Box for Entering Application URL

4. Enter the URL of the application's JAD file.

For information on entering text in the Emulator, see "Inputting Text" on page 49.

5. Click the Go soft button.

The JAM attempts to install the application described by the JAD file.

MIDlet Attributes

This appendix lists and describes the MIDlet attributes, and specifies which attributes go into a suite's manifest and JAD files.

Note – When you work under a development environment, the attributes are automatically placed in the appropriate files. When you use the command line, you must place them manually.

TABLE 7   MIDlet Attributes

Attribute Name	Attribute Description	Attribute File
Required Attributes		
MIDlet-Name	The name of the MIDlet suite that identifies the MIDlets to the user.	JAD and manifest
MIDlet-Version	The version number of the MIDlet suite. The format is `<major>.<minor>.<micro>` as described in the *Java Product Versioning Specification.* It can be used by the application management software for install and upgrade purposes, as well as for communication with the user.	JAD and manifest
MIDlet-Vendor	The organization that provides the MIDlet suite.	JAD and manifest
MIDlet-Jar-URL	The URL from which the JAR file can be loaded.	JAD
MIDlet-Jar-Size	The number of bytes in the JAR file. A development environment should automatically generate this field when the JAR file is built (and prevent it from being edited by the user).	JAD
MicroEdition-Profile	The J2ME profile required, using the same format and value as the system property `microedition.profiles`. For the MIDP 1.0 release the content of this field must be `MIDP-1.0`. In the future, this field will be used to specify the required MIDP version.	manifest

	TABLE 7	MIDlet Attributes

Attribute Name	Attribute Description	Attribute File
`MicroEdition-Configuration`	The J2ME Configuration required using the same format and value as the system property `microedition.configuration`. For CLDC 1.0 compatibility, this field must be `CLDC-1.0`. In the future, this field will be used to specify the required CLDC version.	manifest
Optional Attributes		
`MIDlet-Icon`	The name of a PNG file within the JAR file used to represent the MIDlet suite. It is the icon used by the Java Application Manager to identify the suite.	JAD and/ or manifest
`MIDlet-Description`	The description of the MIDlet suite.	JAD and/ or manifest
`MIDlet-Info-URL`	A URL for information further describing the MIDlet suite.	JAD and/ or manifest
`MIDlet-Data-Size`	The minimum number of bytes of persistent data required by the MIDlet. The device may provide additional storage according to its own policy. The default is zero.	JAD and/ or manifest
`MIDlet-Delete-Confirm`	A text message provided to the user when prompted to confirm deletion of the MIDlet suite.	
`MIDlet-Install-Notify`	The URL to which a POST request is sent to confirm successful installation of this MIDlet suite.	
<User-Defined Attributes>	User-defined attributes relating to specific MIDlets.	JAD
MIDlet-n Attributes		
`MIDlet-<n>`	The name, icon, and class of the nth MIDlet in the JAR file. The lowest value of *<n>* must be 1 and consecutive ordinals must be used. The MIDlet's name identifies it to the user. The MIDlet's icon is specified by the name of a PNG image within the JAR. The MIDlet's class is specified by the name of a class that extends `MIDlet` and has a public no-argument constructor.	manifest

MIDlet Demonstration

The primary purpose of the J2ME Wireless Toolkit is to enable you to develop a MIDlet suite. You can also use it to demonstrate MIDlets for non-development purposes. You can use the J2ME Wireless Toolkit to demonstrate MIDlet suites that are deployed either on a web site or on a local disk without having to perform unnecessary development steps.

Note – If you are not doing actual development with the J2ME Wireless Toolkit, and are only running demonstrations of your MIDlet suite, you are not required to have the J2SE SDK. You can run with only the JRE instead.

Demonstrating MIDlet Suites Deployed on a Local Disk

To demonstrate your application, double-click its JAD file. Alternately, you can use these steps:

1. **From the Windows Start menu, select Programs** -> **J2ME Wireless Toolkit 1.0.4** -> **Run MIDP Application...**

   The Select A JAD File to Run file dialog appears.

2. **Find the JAD file of the application you want to run, and press Run.**

   The Emulator appears.

Demonstrating MIDlet Suites Deployed on a Web Site

The J2ME Wireless Toolkit enables you to execute a MIDlet suite with the toolkit's emulators by visiting the URL of the MIDlet suite's JAD file in a Web browser. The MIDlet suite must be deployed on a Web server.

To deploy a MIDP application on a Web server:

1. **Change the JAD file's MIDlet-Jar-URL property to the URL of the JAR file.**

   This URL must be absolute. For example:

   ```
   MIDlet-Jar-URL: http://mumble.java.sun.com/midlets/example.jar
   ```

2. **Ensure that the Web server recognizes JAD and JAR files:**

   a. **For the JAD file type, set the file extension to** `.jad` **and the MIME type to** `text/vnd.sun.j2me.app-descriptor`**.**

   b. **For the JAR file type, set the file extension to** `.jar` **and the MIME type to** `application/java-archive`**.**

   Note – The details of how to configure a Web server depend on the specific software used.

To run the MIDP application from the Web server:

● **Go to the URL of the JAD file in a Web browser.**

   The Emulator appears.

Internationalization

This appendix discusses setting the language displayed in the J2ME Wireless Toolkit and the localization setting of the emulation environment.

Locale Setting for the Wireless Toolkit

A locale is a geographic or political region or community that shares the same language, customs, or cultural convention. In software, a locale is a collection of files, data, and code, which contains the information necessary to adapt software to a specific geographical location.

Some operations are locale-sensitive and require a specified locale to tailor information for users, such as:

- Messages displayed to the user
- Fonts used or other writing-specific information

By default, all KToolBar strings, that is, the entire User Interface(UI), are displayed in the language of the supported platform's locale.

For example, Japanese characters can be displayed in the KToolBar running on a Japanese Windows NT machine, provided that the correct localized J2ME Wireless Toolkit supplement has been downloaded and installed over the Wireless Toolkit.

You can set the `wtk.locale` property to have the KToolBar displayed in a specified locale's language. For example, you can have the toolkit running on a Japanese Windows NT machine but still have the KToolBar display shown in English by setting the locale property to `en_US`, and making sure that the proper supplement has been downloaded and installed over the J2ME Wireless Toolkit. The `wtk.locale` property should be placed in the *{j2mewtk.dir}*\wtklib\ktools.properties file.

Emulated Locale

The `microedition.locale` property is the MIDP system property that defines the current locale of the device, which is `null` by default. For the J2ME Wireless Toolkit Default Emulator, this value is automatically set to the default locale for the J2SE environment you are running. For example:

- If you are running in an English system in the US, the `microedition.locale` is set to `en_US`.

- If you are running in a French system, the `microedition.locale` is set to `fr_FR`.

For information on `microedition.locale`, see section 4.2, System Properties, in the JSR-37 Mobile Information Device Profile specification at `http://jcp.org/aboutJava/communityprocess/final/jsr037/index.html`.

You can override the default value by adding the `microedition.locale` property to the file {*j2mewtk.dir*}`\wtklib\ktools.properties` file and define the property as desired, as shown in the following examples:

`microedition.locale=en_US`

`microedition.locale=null`

For details on setting a default locale, see the *J2ME Wireless Toolkit Basic Customization Guide.*

Character Encodings

The CLDC system property, `microedition.encoding,` defines the default character encoding name of the device MIDP environment. In the J2ME Wireless Toolkit Default Emulator environment, this property is set according to the underlying window system you are using. The property's value is set to the default encoding for the J2SE environment running on the same window system. For example, in an English window system, the encoding setting is

`microedition.encoding=ISO8859_1`

You can override the default value by adding the `microedition.locale` property to the {*j2mewtk.dir*}`\wtklib\ktools.properties` file. For example, if you want to use UTF-8 as the default setting, you can set the property in the {*j2mewtk.dir*}`\wtklib\ktools.properties` file as follows:

`microedition.encoding=UTF-8`

For more information on character encoding, see section 6.9.2, Property support in the JSR-30 J2ME Connected, Limited Device Configuration specification at

http://jcp.org/aboutJava/communityprocess/final/jsr030/
index.html

Note – All the J2SE encoders are available in the emulated environment. See the *J2ME Wireless Toolkit Basic Customization Guide* for information on how to limit the list of available encoders for a specific device.

Java Compiler Encoding Setting

The `javac.encoding` property determines the encoding used by the javac compiler to compile your source files. The property's value is set to the default encoding for the J2SE environment running on the same window system.

You can override the default value by adding the `javac.encoding` property to the {*j2mewtk.dir*}`\wtklib\ktools.properties` file. For example, if you are running in an English system but find you need to compile a Japanese resource bundle, you can specify a Japanese character set, such as:

`javac.encoding=EUCJIS`

Font Support in the Default Emulator

The default fonts that are used in the emulated environment are set according to the underlying window system locale. By default, the MIDP environment fonts are mapped to the default J2SE environment Java fonts. These fonts usually support all the characters that are required by the current window's locale.

You can override these fonts to support other characters that are not supported by the default fonts. See the *J2ME Wireless Toolkit Basic Customization Guide* for information on how to configure them.

Certificate Manager Utility

This appendix describes the J2ME Wireless Toolkit's certificate manager utility, called MEKeyTool (Mobile Equipment KeyTool). It manages the public keys of certificate authorities (CAs), making it functionally similar to the `keytool` utility that comes with the Java 2 SDK, Standard Edition. The keys can be used to facilitate secure HTTP communication over SSL (HTTPS).

Before using MEKeyTool, you must first have access to a Java Cryptography Extension (JCE) keystore. You can create one using the J2SE `keytool` utility, see `http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html` for more information.

Usage

The MEKeyTool utility is packaged in a JAR file. To run it, open a command prompt, change the current directory to {*j2mewtk.dir*}\bin, and enter the following command:

```
java –jar MEKeyTool.jar <command>
```

Commands

`-import –alias` *<alias>* `[-MEkeystore` *<MEkeystore>*`]` `[-keystore` *<JCEkeystore>*`]` `[-storepass` *<storepass>*`]`

  Import a public key into the given ME keystore from the given JCE keystore using the given JCE keystore password. The default ME keystore is {*j2mewtk.dir*}\appdb_main.ks and the default JCE keystore is {*user.home*}\.keystore.

`-delete` `[-MEkeystore` *<MEkeystore>*`]` `-owner` `<owner>`

  Delete a key from the given ME keystore with the given owner. The default ME keystore is {*j2mewtk.dir*}\appdb_main.ks.

```
-list [-MEkeystore <MEkeystore>]
```

List the keys in the given ME keystore, including the owner and validity period for each. The default ME keystore is {*j2mewtk.dir*}\appdb_main.ks.

```
-help
```

Print the usage instructions for MEKeyTool.

Note – The J2ME Wireless Toolkit contains a default ME keystore called _main.ks, which is located in the appdb\ subdirectory. This keystore includes all the certificates that exist in the default J2SE^{TM} keystore, which comes with the J2SE JDK^{TM} installation.

Index

`wtk.locale` property, 79

X
-Xdebug option, 66
-Xjam option, 69
-Xquery option, 66
-Xrunjdwp option, 67
-Xverbose option, 66