User's Guide

Sun Java™ Wireless Toolkit for CLDC

Version 2.5.2

Sun Microsystems, Inc.
www.sun.com

September 2007

Copyright © 2007 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at http://www.sun.com/patents and one or more additional patents or pending patent applications in the U.S. and in other countries.

U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements. Use is subject to license terms. This distribution may include materials developed by third parties.

Sun, Sun Microsystems, the Sun logo, Java, Javadoc, Java Community Process, JCP, JDK, JRE, J2ME, and J2SE are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

OpenGL is a registered trademark of Silicon Graphics, Inc.

Products covered by and information contained in this service manual are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright © 2007 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. détient les droits de propriété intellectuelle relatifs à la technologie inclus dans le produit qui est décrit dans ce document. En particulier, et ce sans limitation, ces droits de propriété intellectuelle peuvent inclure un ou plus des brevets américains listés à l'adresse http://www.sun.com/patents et un ou les brevets supplémentaires ou les applications de brevet en attente aux Etats - Unis et dans les autres pays. L'utilisation est soumise aux termes de la Licence. Cette distribution peut comprendre des composants développés par des tierces parties.

Sun, Sun Microsystems, le logo Sun, Java, Javadoc, Java Community Process, JCP, JDK, JRE, J2ME, et J2SE sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Ètats-Unis et dans d'autres pays.

OpenGL est une marque déposée de Silicon Graphics, Inc.

Les produits qui font l'objet de ce manuel d'entretien et les informations qu'il contient sont regis par la legislation americaine en matiere de controle des exportations et peuvent etre soumis au droit d'autres pays dans le domaine des exportations et importations. Les utilisations finales, ou utilisateurs finaux, pour des armes nucleaires, des missiles, des armes biologiques et chimiques ou du nucleaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou reexportations vers des pays sous embargo des Ètats-Unis, ou vers des entites figurant sur les listes d'exclusion d'exportation americaines, y compris, et de maniere non exclusive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une facon directe ou indirecte, aux exportations des produits ou des services qui sont regi par la legislation americaine en matiere de controle des exportations et la liste de ressortissants specifiquement designes, sont rigoureusement interdites.

LA DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFACON.

Contents

Preface

This document describes how to work with the Sun Java™ Wireless Toolkit for CLDC.

Who Should Use This Book

This guide is intended for developers creating Mobile Information Device Profile (MIDP) applications with the Sun Java™ Wireless Toolkit for CLDC. This book is not a tutorial in MIDP programming, nor is it a tutorial in programming any of the additional APIs that are supported by the toolkit. You should already understand how to use the Mobile Information Device Profile (MIDP) and the Connected Limited Device Configuration (CLDC).

If you need help getting started with the Java programming language, try the *New to Java Center*:

`http://java.sun.com/learning/new2java/`

For a quick start with MIDP programming, read *Learning Path: Getting Started with MIDP 2.0*:

`http://developers.sun.com/techtopics/mobility/learn/midp/midp20/`

Related Documentation

This section lists related Java Platform, Micro Edition (Java ME) specifications. Java ME was formerly referred to as the Java 2 Platform, Micro Edition, or J2ME™, as you see in some of the specification names. Although specifications are definitive, they are not always the most accessible kind of information. For a variety of developer-centered articles, try Sun's mobility web site:

http://developers.sun.com/techtopics/mobility/

TABLE P-1 Related Documentation

Topic	Title
Customizing the Sun Java™ Wireless Toolkit for CLDC	*Sun Java™ Wireless Toolkit for CLDC Basic Customization Guide*
Release Notes	*Sun Java™ Wireless Toolkit for CLDC Release Notes*
CLDC 1.0 - JSR 30	*J2ME Connected Limited Device Configuration*
MIDP 1.0 - JSR 37	*Mobile Information Device Profile for the J2ME Platform*
PDAP Optional Packages - JSR 75	*PDA Optional Packages for the J2ME Platform*
Bluetooth and OBEX - JSR 82	*Java APIs for Bluetooth*
MIDP 2.1 - JSR 118	*Mobile Information Device Profile 2.0 (Final Release 2 is referred to as MIDP 2.1)*
CLDC 1.1 - JSR 139	*J2ME Connected Limited Device Configuration*
MMAPI - JSR 135	*Mobile Media API*
J2ME Web Services - JSR 172	*J2ME Web Services Specification*
SATSA - JSR 177	*Security and Trust Services APIs for J2ME*
Location API - JSR 179	*Location API for J2ME*
SIP API - JSR 180	*SIP API for J2ME*
Mobile 3D Graphics - JSR 184	*Mobile 3D Graphics API for J2ME*
JTWI - JSR 185	*Java Technology for the Wireless Industry*
WMA 2.0 - JSR 205	*Wireless Messaging API (WMA)*
CHAPI 1.0 - JSR 211	*Content Handler API*
SVG API - JSR 226	*Scalable 2D Vector Graphics API for J2ME*
Payment API - JSR 229	*Payment API*
Advanced Multimedia - JSR 234	*Advanced Multimedia Supplements*

TABLE P-1 Related Documentation (Continued)

Topic	Title
Mobile Internationalization - JSR 238	*Mobile Internationalization API*
Java Binding for OpenGL® ES API - JSR 239	*Java Binding for OpenGL® ES API*
Mobile Service Architecture - JSR 248	*Mobile Service Architecture*

How This Book Is Organized

This guide contains the following chapters and appendices:

Chapter 1 introduces the Sun Java™ Wireless Toolkit for CLDC and the development features it provides.

Chapter 2 describes the development processes for creating and running MIDlets.

Chapter 3 explains how to work with projects in the toolkit. You'll learn how to adjust project properties, manipulate MIDlets, work with the push registry, and understand the project directory structure.

Chapter 4 describes the emulator and explains how to adjust its options and take advantage of its many features.

Chapter 5 shows how you can examine the performance of your application using the method profiler, memory monitor, and network monitor.

Chapter 6 describes how to sign MIDlet suites and manage keys and certificates.

Chapter 7 details support for running and testing wireless messaging applications.

Chapter 8 explains how the Sun Java™ Wireless Toolkit for CLDC supports the Mobile Media API.

Chapter 9 contains information about developing 3D graphics content.

Chapter 10 describes how the toolkit implements access to local files and personal information like contacts and calendar appointments.

Chapter 11 covers the toolkit's Bluetooth and OBEX simulation environment.

Chapter 12 shows how to use the web services stub generator.

Chapter 13 describes how to work with the emulator's location features.

Chapter 14 discusses the toolkit's support for SATSA.

Chapter 15 covers the toolkit's SIP support.

Chapter 16 describes the toolkit's Payment API features.

Chapter 17 shows how to manage resources for the Mobile Internationalization API.

Appendix A describes the application demonstrations that are included in the Sun Java™ Wireless Toolkit for CLDC.

Appendix B explains how to use the functionality of the Sun Java™ Wireless Toolkit for CLDC from the command line.

Appendix C describes localization features in the Sun Java™ Wireless Toolkit for CLDC.

Typographic Conventions

TABLE P-2 describes font uses cases in this document.

TABLE P-2 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*.
{*AaBbCc.dir*}	Variable file names and directories.	In this book, *toolkit* always refers to the directory where the Sun Java™ Wireless Toolkit for CLDC is installed. *workdir* refers to a user's working directory, as explained in Section 1.1.1, "The Installation Directory and the Working Directory" on page 1-1.

Accessing Sun Documentation Online

The following sites provide technical documentation related to Java technology:

- http://developer.sun.com/
- http://java.sun.com/javame/

We Welcome Your Comments

We are interested in improving our documentation. Email your feedback from developers.sun.com.

CHAPTER 1

Introduction

This book describes how to use the Sun Java™ Wireless Toolkit for CLDC.

The Sun Java™ Wireless Toolkit for CLDC is a set of tools that makes it possible to create applications for mobile phones and other wireless devices. Although it is based on the Mobile Information Device Profile (MIDP) 2.1, the Sun Java™ Wireless Toolkit for CLDC also supports many optional packages, making it a widely capable development toolkit.

1.1 About the Multiple User Environment

The Sun Java Wireless Toolkit 2.5.2 for CLDC can be installed on a system running a supported version of either Windows or Linux. All users with an account on the host machine can access the toolkit, either singly or simultaneously.

1.1.1 The Installation Directory and the Working Directory

Windows paths include a drive letter and use backslashes as directory separators. Linux paths use forward slashes. In Linux paths, ~ represents a Linux user's home directory.

To support multiple users the toolkit creates an installation directory that is used as a source for copying. This document uses the variable *workdir* to represent the toolkit working directory and *toolkit* to represent the installation directory. Each user's

personal files are maintained in a separate working directory named `j2mewtk` that has a subdirectory for each version installed. The default location of *workdir* is typically in one of these locations:

Windows:	C:\Documents and Settings*User*\j2mewtk\2.5.2 (where *User* is your account name)
Linux:	~/j2mewtk/2.5.2 (where ~ is your home directory)

Prominent differences between Windows and Linux are described in the documentation. If only one operating system is mentioned, you can assume the above path notation differences apply.

1.1.2 Working Directory Files

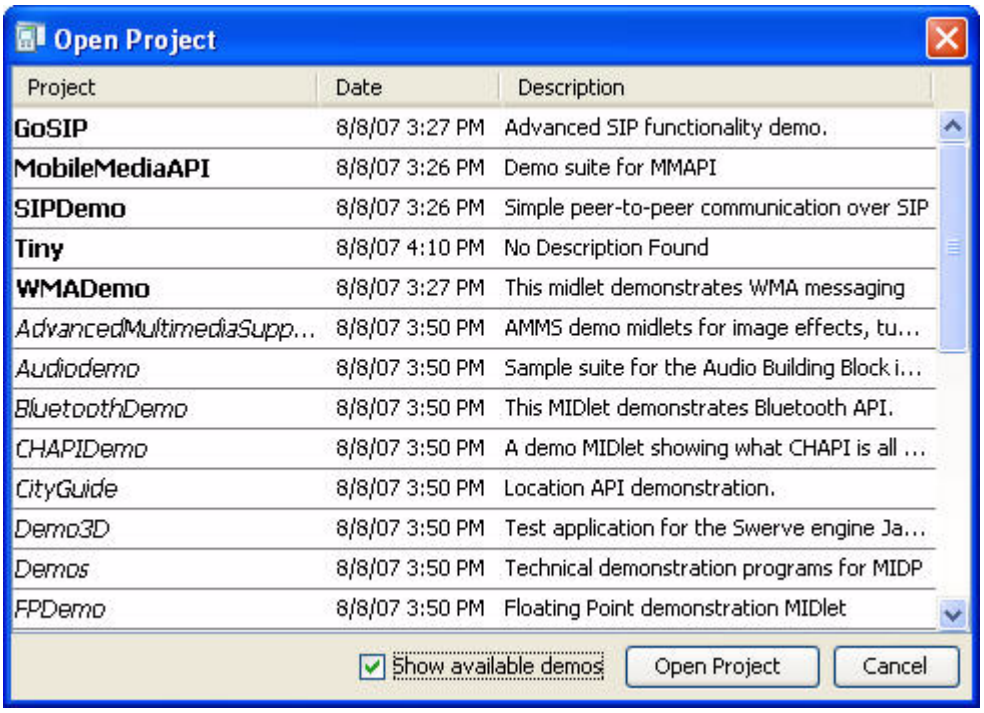
At installation time the installer copies a subset of files from the installation directory to the working directory of each user. The working directory contents are as follows:

- **.settings** - Initially contains `security.properties`.
- **appdb** - The entire applications database is copied to your working directory.
- **apps** - The `apps` directory is created empty in your working directory. If you open a demonstration project it is automatically copied to this location. When you create a new project (Section 2.1, "About Projects" on page 2-1), it is stored here.
- **wtklib** - Contains the emulator properties file (see Chapter 4) and state information for the HTTP and WMA servers and clients.
- **sessions** - This is the default save destination for monitoring tools (see Chapter 5). It is initially empty.

All the source code for the demonstration applications is available in *toolkit*`\apps`, and each demonstration has its own project directory. Any applications placed in *toolkit*`\apps` are visible to all users.

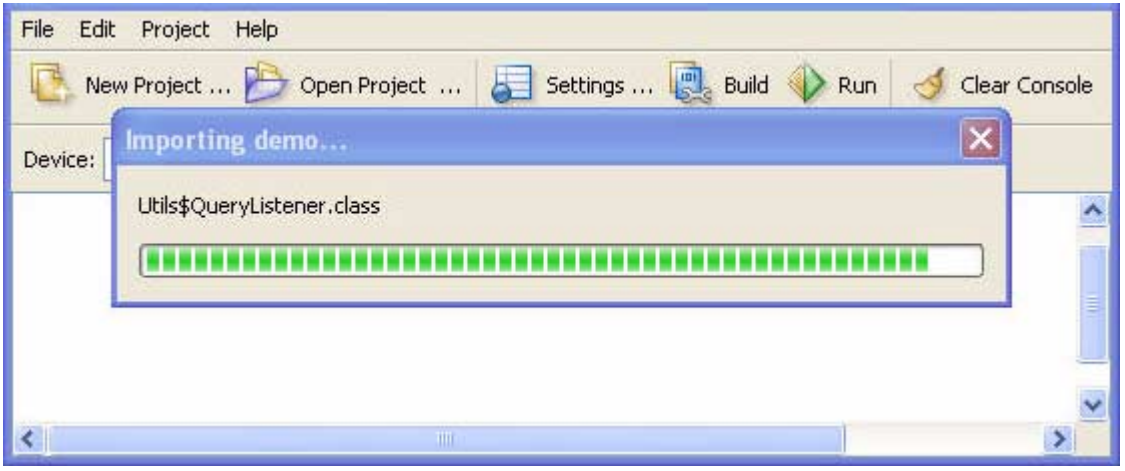
For example, on Windows the source code for the `MobileMediaAPI` demonstration is in *toolkit*`\apps\MobileMediaAPI\src`. When you use the Open Project command, you can see projects in the installation directory and your working directory (if they exist). As shown in FIGURE 1-1, projects in your local directory are shown in **bold** font, and projects in the installation directory are shown in *italic* font. To see only your local files, clear the **Show available demos** check box.

FIGURE 1-1 Project Listing Shows Local Projects in Bold Face



When you open a project from the installation directory, a copy of the application is imported into *workdir*\apps. Often the import process is undetectable, but on slower machines you might see a progress dialog like that in FIGURE 1-2. Any customizations should be made to your local copy.

FIGURE 1-2 Project Import from *toolkit* to *workdir*



1.2 Quick Start

To get started right away, try the demonstration applications that are included with the Sun Java™ Wireless Toolkit for CLDC.

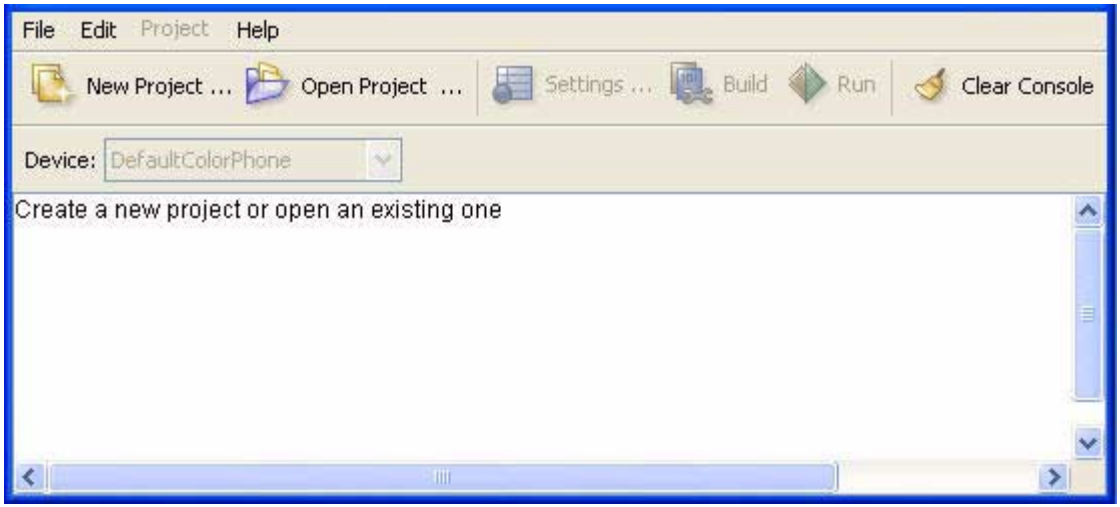
To run the demonstrations, start the toolkit as follows.

- On Microsoft Windows choose Start > Programs > Sun Java Wireless Toolkit 2.5.2 for CLDC > Wireless Toolkit 2.5.2.[1]
- On Linux, change directory to *toolkit*`/bin`. To run the toolkit, enter `./ktoolbar`.

A window similar to FIGURE 1-3 opens.

1. Depending on how Microsoft Windows is configured, you might need to choose Start > All Programs instead of Start > Programs.

FIGURE 1-3 Toolkit User Interface



Click the Open Project button to see a list of all the available applications. As shown in FIGURE 1-1, italicized names denote applications stored in the installation directory, and bold names, if any, are stored in your working directory. Pick one of them and click the Open Project button in the dialog. If you have not opened this project before, a copy will be created in your working directory, as discussed in Section 1.1.2, "Working Directory Files" on page 1-2.

Once the application is open, press the Run button. The emulator appears running the example application.

Most demonstrations are self explanatory, but some have additional instructions. Some demos require you to use Project > Run via OTA instead of run. See Appendix A for general instructions and descriptions of demos.

1.3 Toolkit Components

The Sun Java™ Wireless Toolkit for CLDC has three main components:

- The user interface automates many of the tasks involved in creating MIDP applications.
- The emulator is a simulated mobile phone. It is useful for testing MIDP applications.
- A collection of utilities provides other useful functionality, including a text messaging console and cryptographic utilities.

From the user interface you can build applications, launch the emulator, and start the utilities. Alternatively, the emulator and utilities can be run by themselves, which is useful in many situations. If you want to demonstrate MIDP applications, for example, it's useful to run the emulator by itself.

The only additional tool you need is a text editor for editing source code.

1.4 Toolkit Features

The Sun Java™ Wireless Toolkit for CLDC supports the creation of MIDP applications with the following main features:

- **Building and packaging**: You write the source code and the toolkit takes care of the rest. With the push of a button, the toolkit compiles the source code, preverifies the class files, and packages a MIDlet suite.
- **Running and monitoring**: You can run a MIDlet suite directly in the emulator or install it using a process that resembles application installation on a real device. A memory monitor, network monitor, and method profiler are provided to analyze the operation of your MIDlets.
- **MIDlet suite signing**: The toolkit contains tools for cryptographically signing MIDlet suites. This is useful for testing the operation of MIDlets in different protection domains.

1.5 Toolkit Updates

At installation you are given the option to activate the Check for Product Updates feature. This feature uses the network to check for an update every seven days. To determine whether an update is needed, it collects the following information:

- Operating System
- Country and Language setting on your computer
- Date and Version of the current Wireless Toolkit
- A Unique random user ID generated by the Wireless Toolkit

This data is used to improve the product. Please read Sun's privacy policy at `http://www.sun.com/privacy/index.html`.

To activate or deactivate this feature, select Edit > Preferences and click Network Configuration. Towards the bottom of the panel, check or uncheck the Check for updates box.

1.6 Supported Technology

The Sun Java™ Wireless Toolkit for CLDC supports many standard Application Programming Interfaces (APIs) defined through the Java Community Process™ (JCP™) program. TABLE 1-1 shows the APIs and includes links to the specifications.

TABLE 1-1 Supported JCP Program APIs

JSR API	Name URL
JSR 248 MSA 1.0	*Mobile Service Architecture* `http://jcp.org/en/jsr/detail?id=248`
JSR 185 JTWI 1.0	*Java Technology for the Wireless Industry* `http://jcp.org/en/jsr/detail?id=185`
JSR 139 CLDC 1.1	*Connected Limited Device Configuration* `http://jcp.org/en/jsr/detail?id=139`
JSR 118 MIDP 2.0	*Mobile Information Device Profile* `http://jcp.org/en/jsr/detail?id=118`
JSR 75 PIM and File	*PDA Optional Packages for the J2ME Platform* `http://jcp.org/en/jsr/detail?id=75`
JSR 82 Bluetooth and OBEX	*Java APIs for Bluetooth* `http://jcp.org/en/jsr/detail?id=82`
JSR 135 MMAPI 1.1	*Mobile Media API* `http://jcp.org/en/jsr/detail?id=135`
JSR 172	*J2ME Web Services Specification* `http://jcp.org/en/jsr/detail?id=172`
JSR 177 SATSA	*Security and Trust Services API for Java ME* `http://jcp.org/en/jsr/detail?id=177`
JSR 179 Location	*Location API for Java ME* `http://jcp.org/en/jsr/detail?id=179`
JSR 180 SIP	*SIP API for Java ME* `http://jcp.org/en/jsr/detail?id=180`
JSR 184 3D Graphics	*Mobile 3D Graphics API for J2ME* `http://jcp.org/en/jsr/detail?id=184`
JSR 205 WMA 2.0	*Wireless Messaging API* `http://jcp.org/en/jsr/detail?id=205`

TABLE 1-1 Supported JCP Program APIs *(Continued)*

JSR API	Name URL
JSR 211 CHAPI	*Content Handler API* `http://jcp.org/en/jsr/detail?id=211`
JSR 226	*Scalable 2D Vector Graphics API for J2ME* `http://jcp.org/en/jsr/detail?id=226`
JSR 229	*Payment API* `http://jcp.org/en/jsr/detail?id=229`
JSR 234 AMMS	*Advanced Multimedia Supplements* `http://jcp.org/en/jsr/detail?id=234`
JSR 238 MIA	*Mobile Internationalization API* `http://jcp.org/en/jsr/detail?id=238`
JSR 239	*Java Binding for OpenGL® ES API* `http://jcp.org/en/jsr/detail?id=239`
JSR 248	*Mobile Service Architecture* `http://jcp.org/en/jsr/detail?id=248`

CHAPTER 2

Developing MIDlet Suites

This chapter describes how you can use the Sun Java™ Wireless Toolkit for CLDC to create applications. It begins with a description of toolkit projects, then works through the development process.

You are likely to follow two basic development cycles in creating MIDlet suite applications. The first is quicker and simpler. It is useful in your initial development. The second cycle is longer but allows for more comprehensive and realistic testing.

The end of the chapter contains information on how to use the Sun Java™ Wireless Toolkit for CLDC with advanced development tools like an obfuscator and a debugger. A final section briefly describes how to configure a web server to serve MIDP applications.

2.1 About Projects

In the Sun Java™ Wireless Toolkit for CLDC, MIDlet suites are organized into *projects*, where the end result of one project is one MIDlet suite. A project contains all of the files that will be used to build a MIDlet suite, including Java source files, resource files, and the MIDlet descriptor.

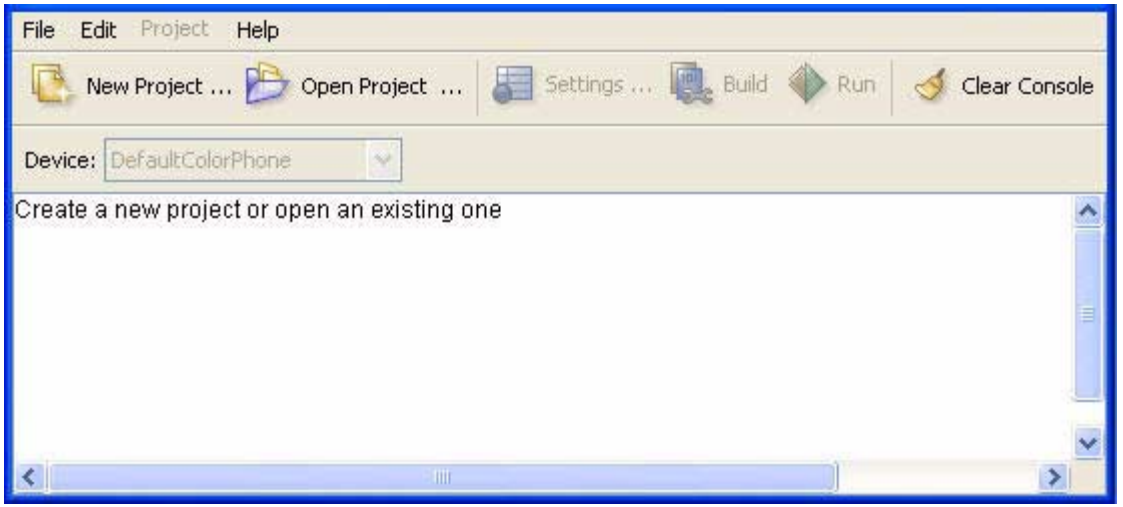
The Sun Java™ Wireless Toolkit for CLDC works on one project at a time. You can create a new project or open an existing project.

In this chapter uses a very simple example project. As you read about each step in the development cycles, you can work along in the toolkit.

To create a new project, first start the user interface. On Microsoft Windows, choose Start > Programs > Sun Java Wireless Toolkit 2.5.2 for CLDC > Wireless Toolkit 2.5.2.[1] The user interface appears, as shown in FIGURE 2-1.

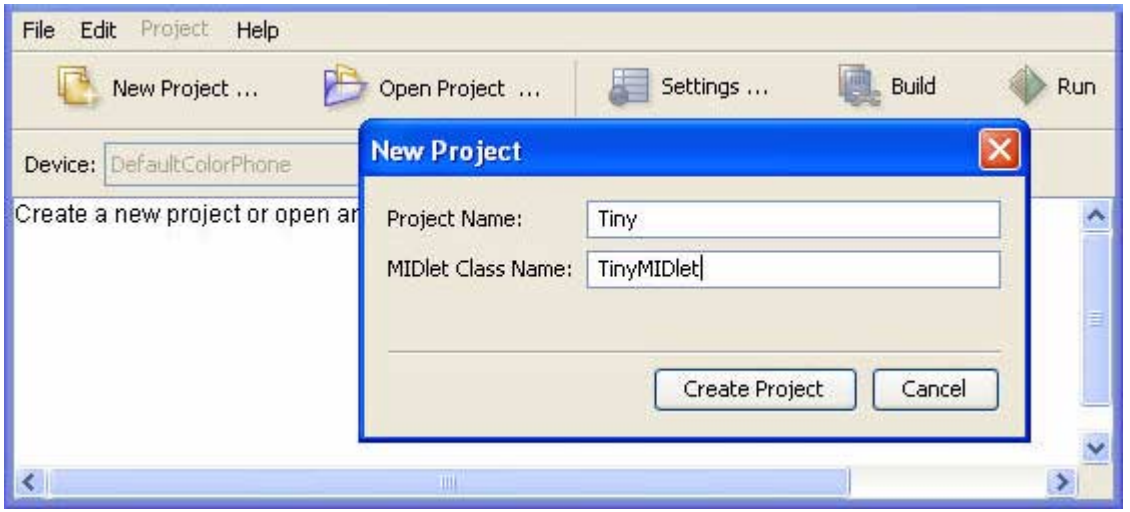
1. Depending on how Microsoft Windows is configured, you might need to choose Start > All Programs instead of Start > Programs.

FIGURE 2-1 The Toolkit's User Interface



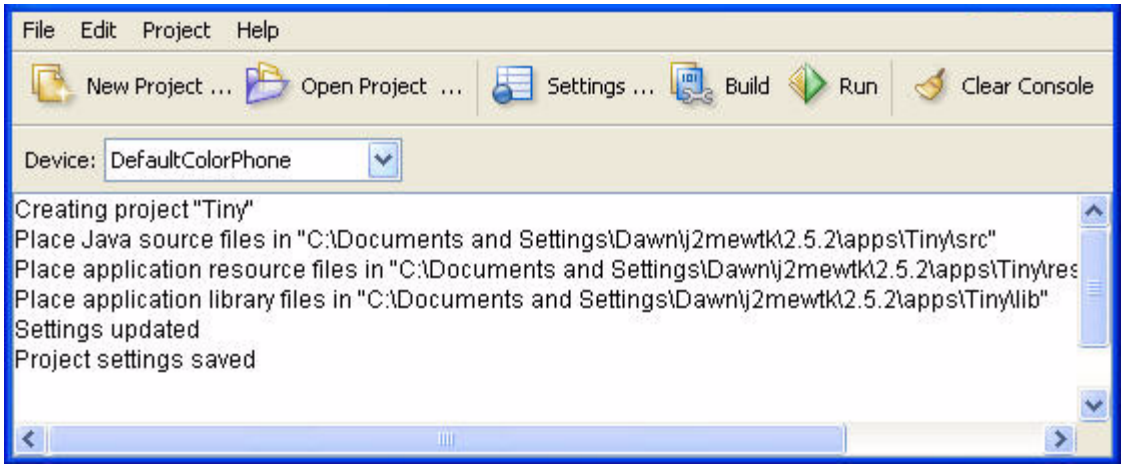
Click New Project. The toolkit will ask you for the name of the project and the name of the MIDlet class you will write. Fill in the names and click Create Project.

FIGURE 2-2 Creating a New Project



The Settings window is displayed. Your choices affect the build environment for the project. The default options are fine for this example, so click OK to dismiss the window. Messages appear in the console telling you exactly where to store the source code and resource files for this project.

FIGURE 2-3 File Locations in the Console



2.2 Simple Development Cycle

The simple development cycle looks like this:

Edit source code > Build > Run

1. **Edit source code.**

   In this step, you create Java source files and resource files that will be used by your application.

2. **Build.**

   The toolkit compiles and preverifies your Java source files.

3. **Run.**

   The compiled Java class files are run on the emulator.

If an error occurs when the toolkit attempts to compile your source files, go back and edit them again. If you find a bug when you are testing your application in the emulator, edit the source files to fix the bug.

Now that you understand the simple development cycle at a high level, the rest of this section illustrates how you can accomplish each step using the Sun Java™ Wireless Toolkit for CLDC.

2.2.1 Edit Source Code

Editing source code is the only step in which the Sun Java™ Wireless Toolkit for CLDC is no help at all. Use the text editor of your choice to create and edit source code files. If you don't have a favorite text editor, try jEdit, at http://jedit.org/.

If you are following along with the sample project, create a new Java technology source file `TinyMIDlet.java`. Save it in your project's source directory. For example, on Windows this would be *workdir*`\apps\Tiny\src\TinyMIDlet.java`. This file contains this simple MIDlet:

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;

public class TinyMIDlet
    extends MIDlet
    implements CommandListener {
  public void startApp() {
    Display display = Display.getDisplay(this);

    Form mainForm = new Form("TinyMIDlet");
    mainForm.append("Welcome to the world of MIDlets!");

    Command exitCommand = new Command("Exit", Command.EXIT, 0);
    mainForm.addCommand(exitCommand);
    mainForm.setCommandListener(this);

    display.setCurrent(mainForm);
  }
  public void pauseApp () {}

  public void destroyApp(boolean unconditional) {}

  public void commandAction(Command c, Displayable s) {
    if (c.getCommandType() == Command.EXIT)
      notifyDestroyed();
  }
}
```

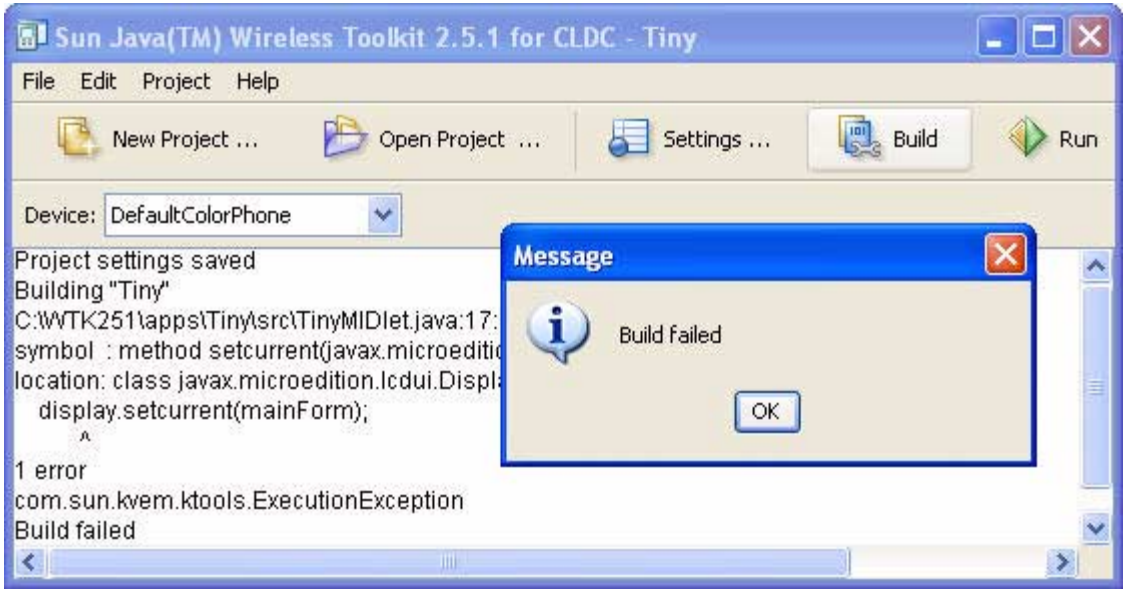
Save the file when you are finished.

2.2.2 Build

The next step is to build your source code. The toolkit makes this part very easy.

In the user interface, click the Build button. Assuming you saved your source file in the right place, the toolkit finds it and compiles it. Compilation errors are displayed in the console. If you have errors, as shown in FIGURE 2-4, edit the source code to fix them. Once you eliminate your errors, the console informs you that the build has completed successfully.

FIGURE 2-4 Messages About Building

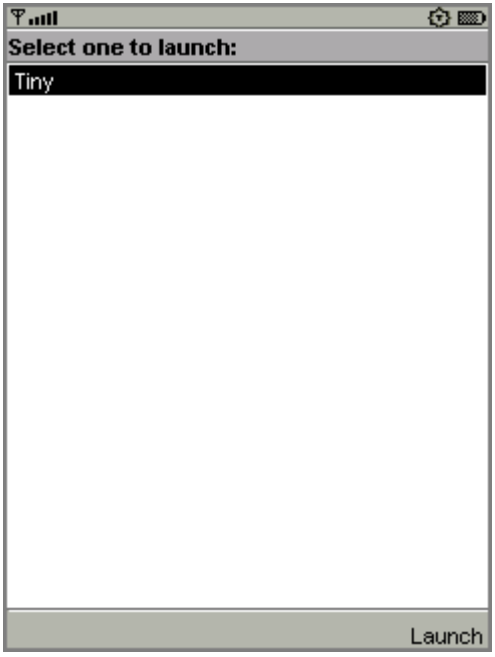


Behind the scenes, the toolkit also preverifies the compiled class files. MIDlet class files must be preverified before they can be run on a MIDP device or emulator. The toolkit silently handles this detail for you. See the CLDC specification for more information on preverification.

2.2.3 Run

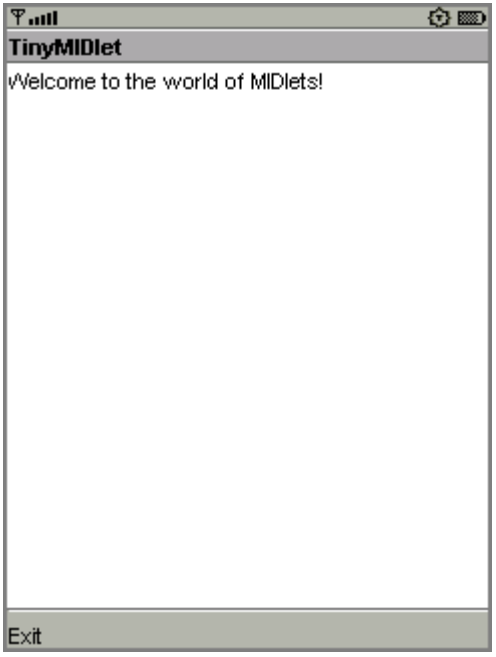
Once the project builds successfully, you are ready to try it out in the emulator. Click the Run button. The emulator displays a list of all the MIDlets in your project.

FIGURE 2-5 List of Project MIDlets



Choose the MIDlet you want and select Launch. If you're following along with the `TinyMIDlet` example, the result is as shown in FIGURE 2-6.

FIGURE 2-6 `TinyMIDlet` in Action



2.3 Full Development Cycle

The second development cycle is slightly more complicated. It consists of the following high-level steps:

Edit source code > Package > Install > Run

1. **Edit source code.**

   This is the same as in the simple cycle.

2. **Package.**

   In this step, the Sun Java™ Wireless Toolkit for CLDC compiles and preverifies the source files (essentially the same as the Build step from before). Then it bundles the Java class files and resource files into a MIDlet suite Java Archive (JAR) file and a MIDlet suite descriptor.

3. **Install.**

   MIDlet suites need to be installed before they can be run. You can install the MIDlet suite into the Sun Java™ Wireless Toolkit for CLDC emulator or a real device.

4. **Run.**

   As in the simple development cycle, run your application and test for bugs.

In the full development cycle, the first step is identical to the simple development cycle. Editing source code is the same as always. The Build step is now incorporated in packaging.

The full development cycle includes two new steps, packaging and installing. Finally, running an installed application is different in important ways from running an application in the simple development cycle.

2.3.1 Package

The Sun Java™ Wireless Toolkit for CLDC automates the task of packaging a MIDlet suite. The end result of packaging is two files, a MIDlet descriptor and a MIDlet suite JAR file. The descriptor is a small text file that contains information about the MIDlet suite. The JAR file contains the class files and resources that make up the MIDlet suite. Devices can use the descriptor to learn about the application before downloading the entire JAR file, an important consideration in a memory-lean, bandwidth-starved wireless world.

To ask the toolkit to package your MIDlet suite, choose Project > Package > Create Package. The MIDlet suite descriptor and JAR file are generated and placed in the `bin` directory of your project.

Packaging might involve additional steps. You can use a code obfuscator to shrink the size of the MIDlet suite JAR file, a technique that is described later in this chapter. In addition, the toolkit provides tools to enable you to cryptographically sign MIDlet suites. See Chapter 6 for more information.

2.3.2 Install

To properly test a MIDlet suite, install it into the toolkit's emulator or a real device. When you press the Run button in the user interface, the MIDlet suite is not installed into the emulator. Instead, the emulator runs the MIDlet classes directly.

The emulator also can install applications into its memory in a process that resembles how applications are transmitted and installed Over the Air (OTA) on real devices. To install applications in the Sun Java™ Wireless Toolkit for CLDC emulator, choose Project > Run via OTA.

The emulator window opens, but instead of running your MIDlet classes directly, the emulator shows the Application Management Software (AMS) welcome screen. The emulator's software is an example of the type of software that real devices must have to manage MIDlet suites.

FIGURE 2-7 Emulator AMS Welcome Screen

Choose Apps to go to the main list of installed applications. Select Install Application and press the select button on the emulator. The emulator prompts you for the URL location of the application you want to install. The URL is already completed for you.

FIGURE 2-8 URL Prompt

From the menu, choose Go to begin the installation. The emulator shows a list of the applications it finds at the URL. Choose the only one and select Install from the menu. The emulator gives you one last chance to confirm your intentions.

FIGURE 2-9 Confirming the Installation

Choose Install again to finish the installation. You are returned to the emulator's installed list of applications, which now includes the application you just installed.

FIGURE 2-10 Application Menu

Run via OTA is an extremely useful mechanism that makes it easy to install your MIDlet suite on the toolkit emulator. Some features *must* be tested using this technique, including the push registry and the installation of signed MIDlet suites.

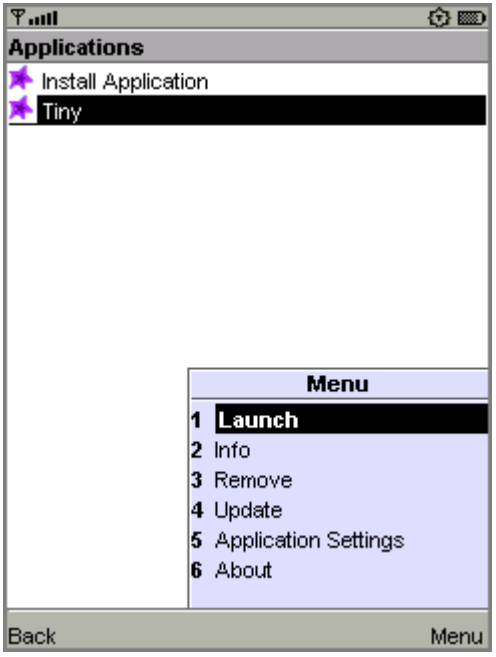
If you want to test your MIDlet suite on a real device, you must install it first. How this happens depends heavily on the device you are using. The following two possibilities are most likely:

- You can deploy the application on a web server, then transmit the application from server to device using the Over the Air (OTA) protocol described in the MIDP 2.0 specification. This is most likely the same mechanism that users will encounter when they go to purchase or install your application.
- You might be able to transfer the MIDlet suite to the device using a Bluetooth, infrared, or serial connection. This is quite a bit simpler than running a web server, and although it won't give you any insights into the process of installing your application on the device using OTA, it enables you to see how your application performs on the device.

2.3.3 Run

Once the application is installed, choose the application from the list and choose Launch from the menu.

FIGURE 2-11 Launching the Installed Application



Running an application on a real device depends heavily on the device itself. Consult your device documentation for information.

2.4 Creating a Project from a MIDlet Suite

You can also create a Sun Java™ Wireless Toolkit for CLDC project from a MIDlet suite archive (`.jar` file) and descriptor (`.jad` file). This is useful for running MIDlet suites from the toolkit user interface even when you don't have the source code available. You can use the toolkit to easily manipulate the attributes in the descriptor, or you can run the project and use the toolkit's monitoring tools (described in Chapter 5) to scrutinize its behavior.

To create a project based on a MIDlet suite choose File > Create project from JAD/JAR. Navigate to the descriptor you wish to use and click Open. Note that the descriptor and the JAR file must be in the same directory.

2.5 Using an Obfuscator

An *obfuscator* is a tool that reduces the size of class files. MIDlet suites need to be compact, both to minimize download times and to comply with sometimes stringent limits on JAR file size imposed by manufacturers or carriers. Using an obfuscator is one way (not the only way) that you can keep your MIDlet suite JAR file small.

You can use an obfuscator in the packaging step of the development cycle. Although the Sun Java™ Wireless Toolkit for CLDC doesn't come with an obfuscator, it is already configured to use the ProGuard obfuscator. All you need to do is download ProGuard and put it in a place where the toolkit can find it.

ProGuard is published under the terms of the GNU General Public License (GPL). If you are comfortable with the terms of the license, you can download and use ProGuard free of charge.

2.5.1 Installing ProGuard

Follow these steps to install ProGuard:

1. **Go to the ProGuard web site,** `http://proguard.sourceforge.net/`**.**
2. **Download the latest version.**
3. **Uncompress the** `proguard.jar` **file from the** `lib` **directory of the ProGuard installation to the** *toolkit*`/bin` **directory.**

2.5.2 Using ProGuard

Once ProGuard is installed, you can use it by choosing Project > Package > Create Obfuscated Package.

In some cases, you need to provide a script file that controls how the obfuscator works. If you are loading classes using `Class.forName()`, for example, you need to tell ProGuard to leave the class names unchanged.

To call the script file you must be able to modify your own copy of the file `ktools.properties`.

- **Copy** *toolkit*`/wtklib/`*os*`/ktools.properties` **to** *workdir*`/wtklib/`*os*`/ktools.properties`**.**

Create a script file using a text editor, then save it under the project's main directory. Consult the ProGuard documentation for information on script files. Next, tell the toolkit how to find this file. To do this, edit the `ktools.properties` file you have copied to *workdir*`/wtklib`. Add a line as follows:

`obfuscate.script.name:` *scriptfile*

Replace *scriptfile* with the name you used for the script file. You must quit and restart the toolkit for the change to take effect.

2.6 Using a Debugger

A variation on running your application is running it with a debugger. A debugger enables you to monitor the running application more closely, set breakpoints, and examine variables.

You must supply your own debugger. You can use the `jdb` debugger from the Java SE platform or another debugger of your choice. If you want to use a debugger, an Integrated Development Environment (IDE) like Sun Java Studio Mobility software, which incorporates the Sun Java™ Wireless Toolkit for CLDC, is your most likely choice. See `http://www.sun.com/software/products/jsmobility/` for more information.

Begin by choosing Project > Debug. Enter the TCP/IP port number that the debugger uses to connect to the emulator. Click Debug. The emulator begins running and waits for a connection from a debugger.

Start your debugger and attach it to the port you specified. 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.

Debugging MIDlets has information about using `jdb` with the Sun Java™ Wireless Toolkit for CLDC. It is available at `http://developers.sun.com/techtopics/mobility/midp/questions/jdb/`.

2.7 Deploying Applications on a Web Server

The MIDP 2.0 specification includes the *Over The Air User Initiated Provisioning Specification*, which describes how MIDlet suites can be transferred over-the-air (OTA) to a device. You can test this type of scenario using the Sun Java™ Wireless Toolkit for CLDC emulator.

To deploy a packaged MIDP application remotely on a web server, change the Java Application Descriptor (JAD) file's `MIDlet-Jar-URL property` to the URL of the JAR file. The URL must be an absolute path. For example:

```
MIDlet-Jar-URL: http://your.server.com/midlets/example.jar
```

Next, ensure that the web server uses the correct MIME types for JAD and JAR files:

- For MIDlet suite descriptors, map the `.jad` extension to the `text/vnd.sun.j2me.app-descriptor` MIME type.
- For MIDlet suite JAR files, map the `.jar` extension to the `application/java-archive` MIME type.

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

The emulator implements the device behavior during OTA provisioning. You can use the emulator to test and demonstrate the full provisioning process of MIDlet suites from a server to the device. All you need to do is launch the emulator's AMS. You might already be familiar with the AMS if you have used the Run via OTA option.

To launch the emulator's AMS, you have two options:

- In the Microsoft Windows start menu, choose Start > Programs > Sun Java Wireless Toolkit 2.5.2 for CLDC > OTA Provisioning.
- From the command line, issue the following command:

Windows:	*toolkit*`\bin\emulator -Xjam`
Linux:	*toolkit*`/bin/emulator -Xjam`

Now follow the AMS prompts to install your application. This process is very similar to the Run via OTA option described earlier in this chapter, except you must enter the URL of your own server to install your application.

CHAPTER 3

Working With Projects

In the last chapter, you learned how the Sun Java™ Wireless Toolkit for CLDC helps you with the MIDP development cycle. This chapter delves more deeply into the details of working with projects, including the following:

- Selecting the target APIs for a project
- Manipulating MIDlet suite attributes, including the list of MIDlets
- Understanding the project directory structure
- Including third-party libraries in a project

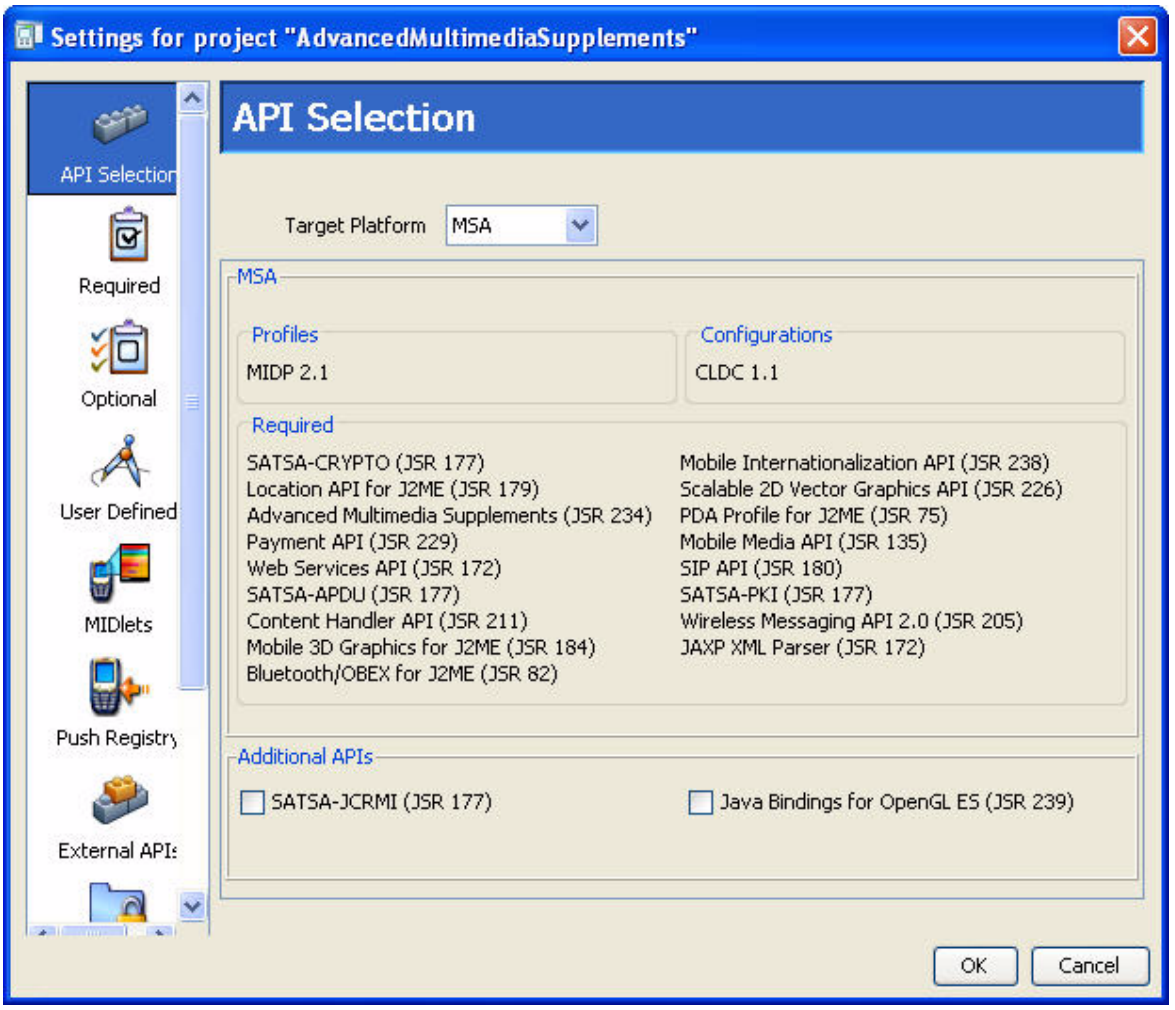
3.1 Selecting APIs

Each project is built against some set of APIs. The Sun Java™ Wireless Toolkit for CLDC supports many APIs. The full list is detailed in Section 1.6, "Supported Technology" on page 1-7. The toolkit enables you to develop applications for some subset of APIs based on the type of devices you expect to run your software.

For example, even though the toolkit supports JSR 184, the Mobile 3D Graphics API, you might want to develop applications that don't make use of that API. The project's API Selection settings make it possible to choose only the APIs you want to use.

To see how this works, launch the toolkit and open a project. Click Settings to bring up the window shown in FIGURE 3-1:

FIGURE 3-1 Project Settings Window



On the API Selection pane, the Target Platform setting controls the appearance of the rest of the pane. Choose the setting that best suits your need, and tweak your selection with the controls below. For example, if you're developing applications for devices that are compliant with the Java Technology for the Wireless Industry JSR, choose JTWI from the combo box. Then use the controls below to specify a version of CLDC and choose optional APIs.

The toolkit applies your selections when you compile your source code.

Note – API selections do not apply to the emulator. The emulator always supports all the available APIs. The API selections you make in the project settings apply only to building a project. In essence, the API selections choose which classpath the toolkit uses for compiling and preverifying your source files.

3.2 Changing MIDlet Suite Attributes

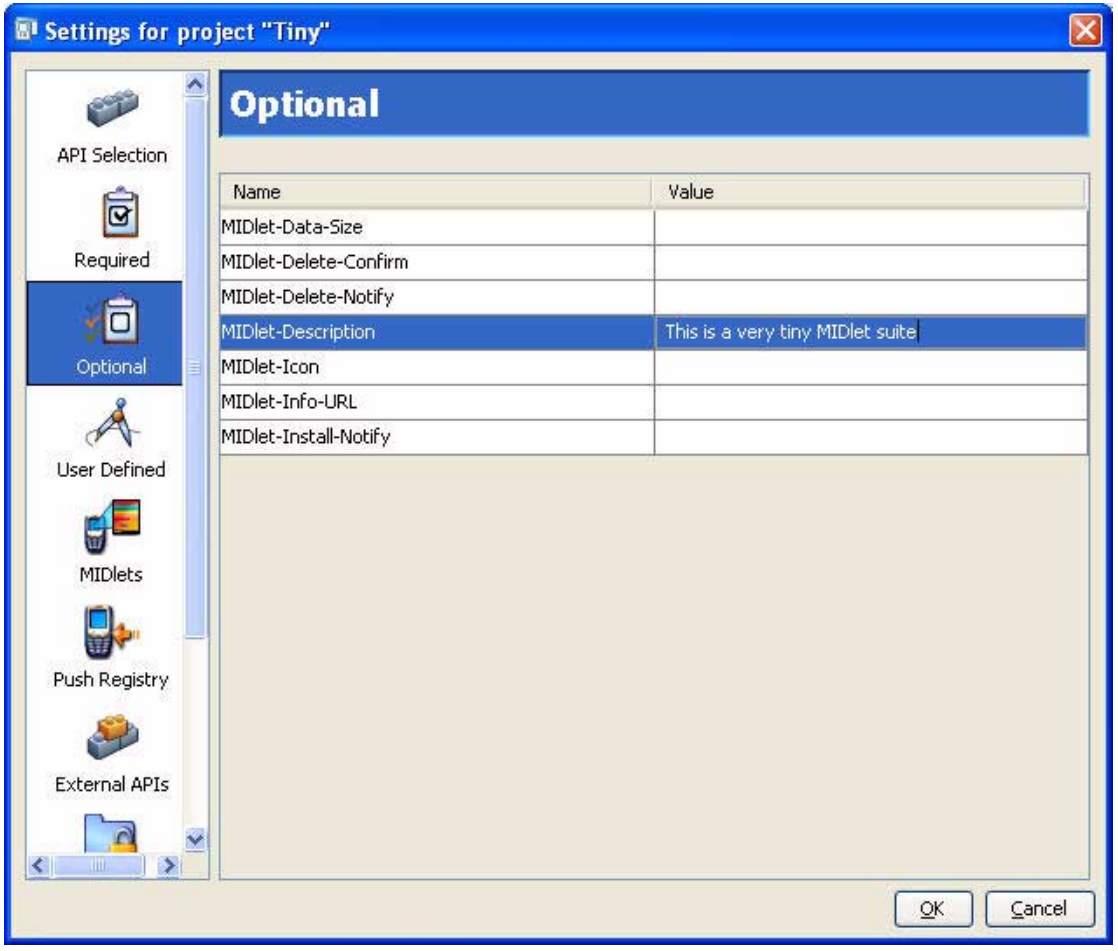
The project settings window also allows you to control the MIDlet suite attributes, which are stored in the descriptor as well as the manifest file of the MIDlet suite JAR file.

To see the attributes, open a project, then click the Settings button. The icon bar on the left of the settings window has three icons for attributes: Required, Optional, and User Defined.

Consult the MIDP 2.0 specification, final release 2 (referred to as MIDP 2.1) for the definitions of the required and optional attributes. The Sun Java™ Wireless Toolkit for CLDC takes care of most of the details. In the early stages of development, you might not have to worry about the attributes at all. Once your application is stable and you're starting to think about deploying on real devices and going to market, adjust the values.

To adjust a value on the Required or Optional panes, click in the cell next to the attribute key you wish to change. Type in the new value.

FIGURE 3-2 Editing MIDlet Suite Attributes



To create new user-defined attributes, click the User Defined icon. Click the Add button and fill in the property name and value, then click OK.

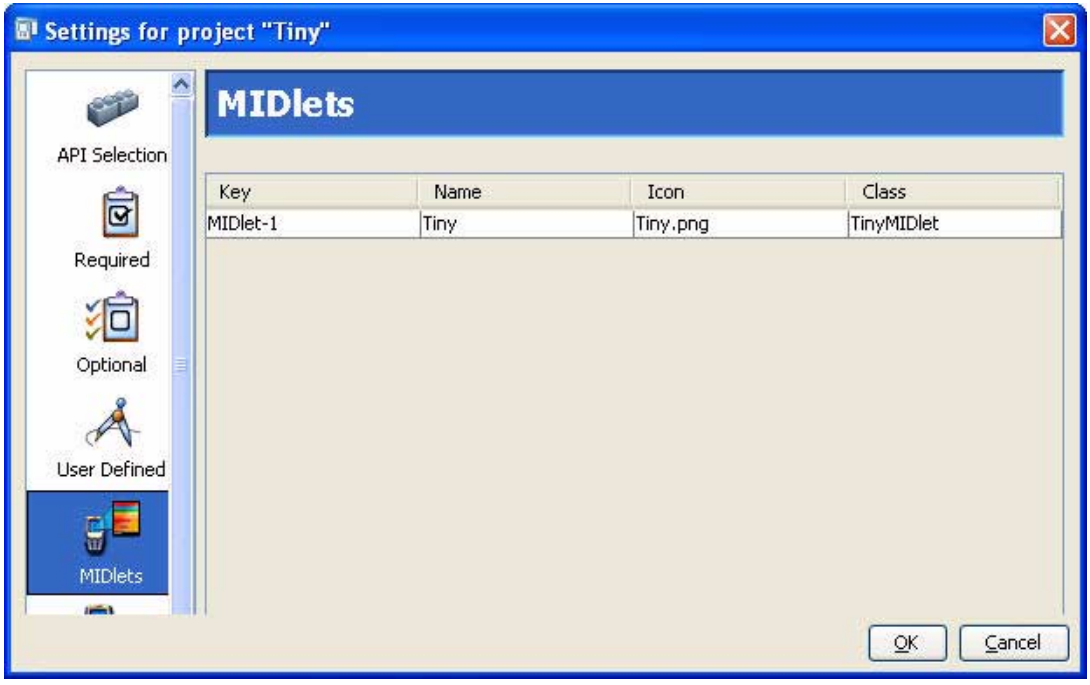
To edit the user-defined property value, click the value column next to the key, just as you would with required or optional attributes.

To remove an attribute, select an attribute and click Remove.

3.3 Manipulating MIDlets

The project settings also provide a way to add or modify the MIDlets that are contained in the current MIDlet suite project. To see how this works, start the toolkit and open an existing project. Click Settings and choose the MIDlets icon. You will see a list of all MIDlets in the project. If you just created a new project, the toolkit automatically fills in the first MIDlet entry.

FIGURE 3-3 List of MIDlets in a Project



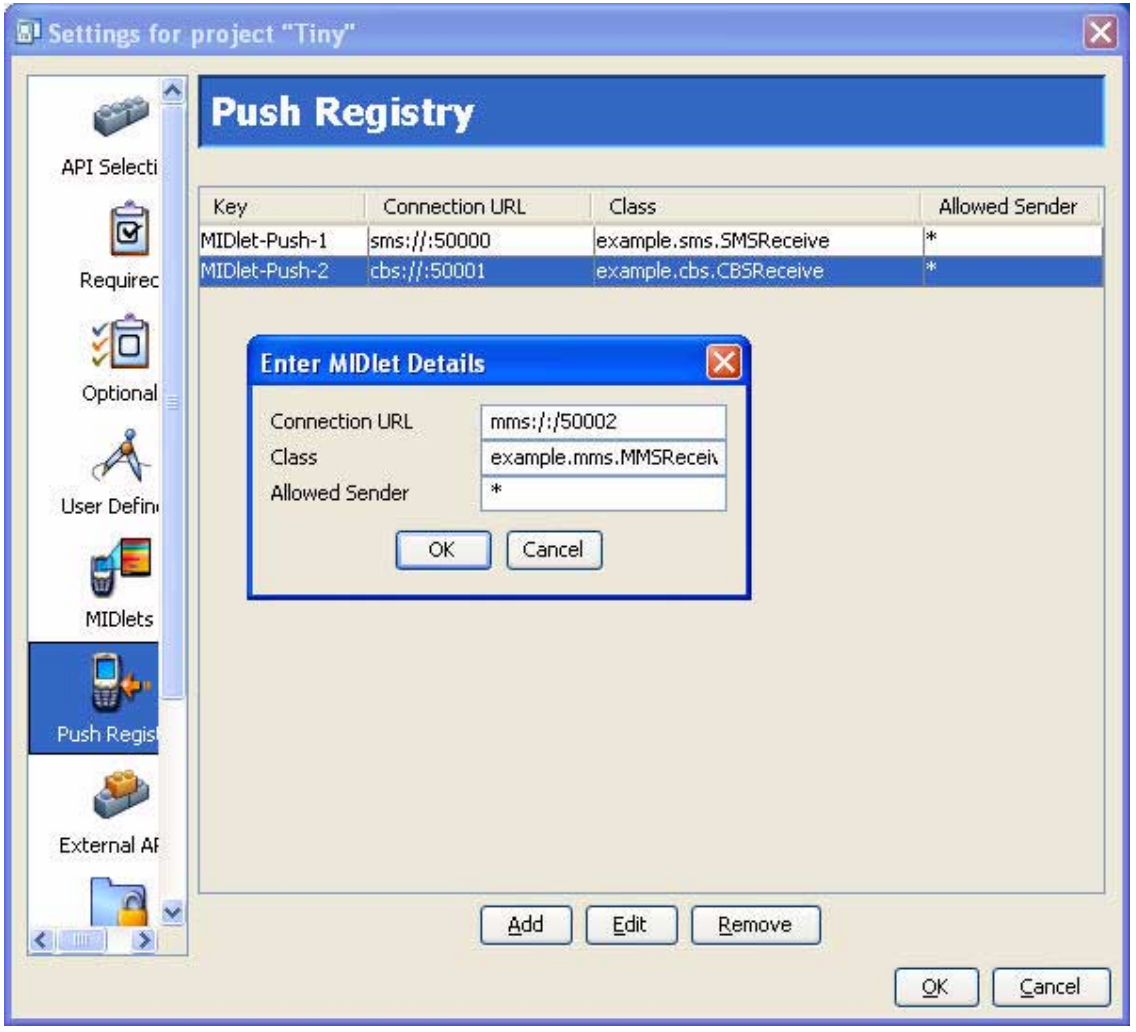
To add a new MIDlet, click Add. Fill in the name, icon file name, and class name. You can leave the icon file name blank if you wish. To change values or remove MIDlet entries, use the Edit and Remove buttons.

The MIDlet names are presented to the user in the order shown when the MIDlet suite is launched. You can modify the order by selecting a MIDlet and clicking Move Up or Move Down.

3.4 Using the Push Registry

You can also use project settings to work with a MIDlet suite's push registry settings. Click on Settings... and choose the Push Registry icon.

FIGURE 3-4 Project Push Registry Settings



To add an entry to the push registry, press Add and fill in values for the connection URL, MIDlet class, and allowed sender, then click OK. To edit an entry, select the entry and press the Edit button. To remove a push registry entry, select it and click Remove.

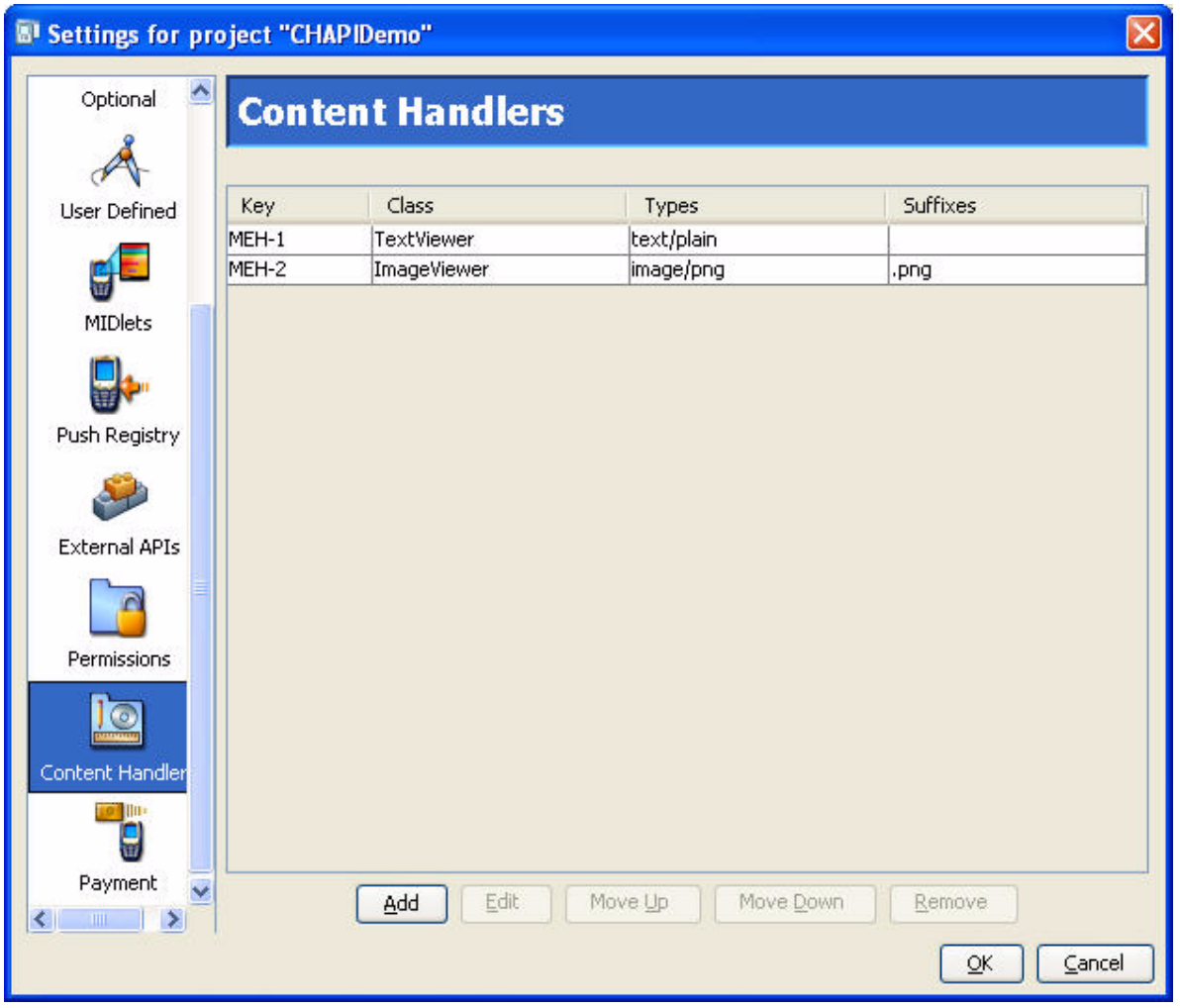
If you do make push registry entries for your application, make sure you also enter the appropriate permissions. See Chapter 6 for details.

3.5 Setting Up Content Handlers

The Sun Java™ Wireless Toolkit for CLDC supports the Content Handler API (CHAPI), which is defined by JSR 211. The basic concept of CHAPI is that MIDlets can be launched in response to incoming content (files). Modern mobile phones can receive content using SMS, infrared, Bluetooth, e-mail, and other methods. Most content has an associated content type. CHAPI specifies a system by which MIDlets can be launched in response to specific types of content.

To modify the content handler settings in your project, click on Settings and choose the Content Handlers pane.

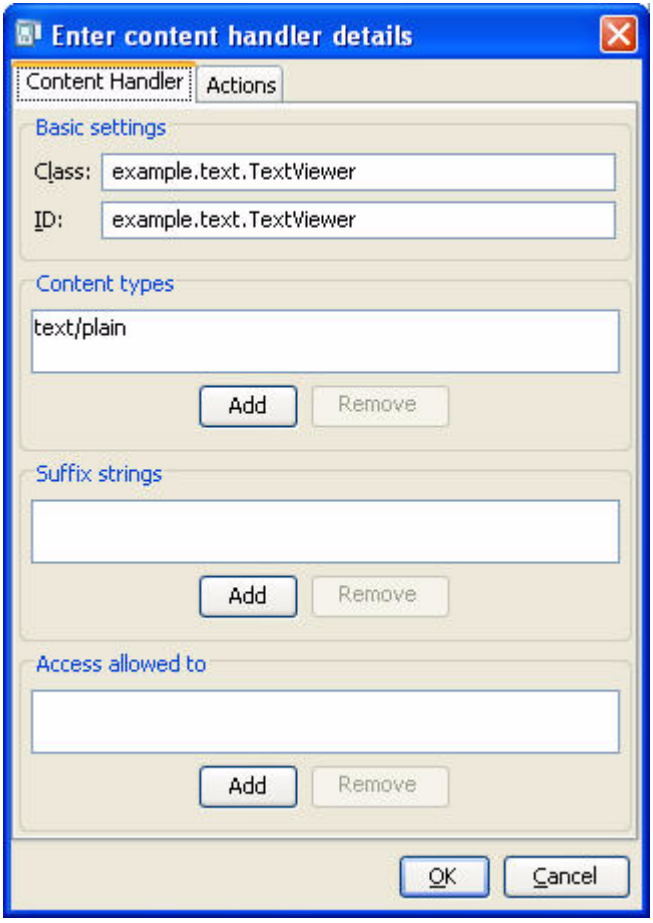
FIGURE 3-5 Configuring Content Handlers



Each line in the list represents the settings for a content handler. In this example, two content handlers have been configured, one for `TextViewer` and one for `ImageViewer`. To create a new content handler, press Add, or to edit an existing content handler, press Edit. You can adjust the order of the content handlers by selecting one and using the Move Up and Move Down buttons. To remove a content handler from the list, press Remove.

When you add or edit a content handler, the details window appears:

FIGURE 3-6 Content Handler Details

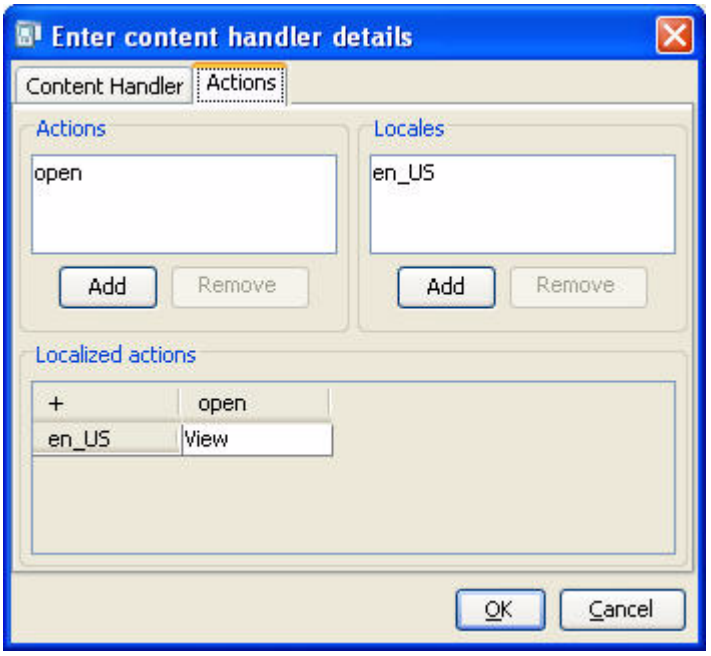


In the Class field, fill in the name of a MIDlet. ID is an identification string that can be used to invoke a content handler and control access.

Content types is a list of content types for which this content handler is responsible. Use Add and Remove to manage the list. *Suffix strings* is a list of URL suffixes that are often a substitute for an explicit content type. Finally, *Access allowed to* is a list of content handler IDs that indicates which other content handlers have access to this content handler. If the list is empty, access to this content handler is granted to every other content handler.

Content handlers have associated actions, which give invoking applications a choice about how to handle content. An image viewer content handler, for example, might include an action for viewing the image at its original size and another action that makes the image fill the available screen space. Click the Actions tab of the content handler details window to edit the actions for a content handler.

FIGURE 3-7 Content Handler Actions



The Actions list contains the internal names of the actions for this content handler. Locales is a list of all the locales for which human-readable action names will be provided. Localized actions is a grid which contains the human-readable action names for various locales. Each locale is represented by a row, while the actions are listed as columns. You can see all the human-readable action names for a particular locale by reading across a single row.

3.6 Project Directory Structure

Projects have a standard directory structure. The project itself is represented by a directory in the `apps` subdirectory. For example, on Windows the `demos` project resides in *workdir*`\apps\demos`. The following table describes the project directory structure.

TABLE 3-1 Project Directory Structure

Directory	Description
`bin`	The MIDlet suite descriptor and JAR file are placed in this directory when you package the project. This directory also contains the unpackaged manifest information and might include an HTML file that is used internally if you use Run via OTA.
`classes`	This directory is used to store compiled class files.
`lib`	Place a third-party library in this directory to include it in this project.
`res`	Place images, sounds, and other resource files in this directory. They are packaged into the root of the MIDlet suite JAR file.
`src`	Place source files in this directory.
`tmpclasses`	For toolkit use.
`tmplib`	For toolkit use.

In addition, the project directory contains a `project.properties` file that contains information about the project.

To remove temporary directories and files from the project, choose Project > Clean.

3.7 Using Third-Party Libraries

The Sun Java™ Wireless Toolkit for CLDC enables you to incorporate third-party libraries in your applications. Using third-party libraries can reduce your development time by providing functionality you don't wish to build yourself, but keep a close eye on the size of your MIDlet suite JAR file.

When you use a third-party library in your application, your JAR expands by the size of the third-party library. You can use an obfuscator to reduce the code size, and a good obfuscator even eliminates whatever parts of the library you are not using. Even with the use of an obfuscator, a third-party library is probably still larger than

your own custom code, carefully written from scratch. You have to evaluate the trade-off between reducing your development time and the size of your MIDlet suite JAR file.

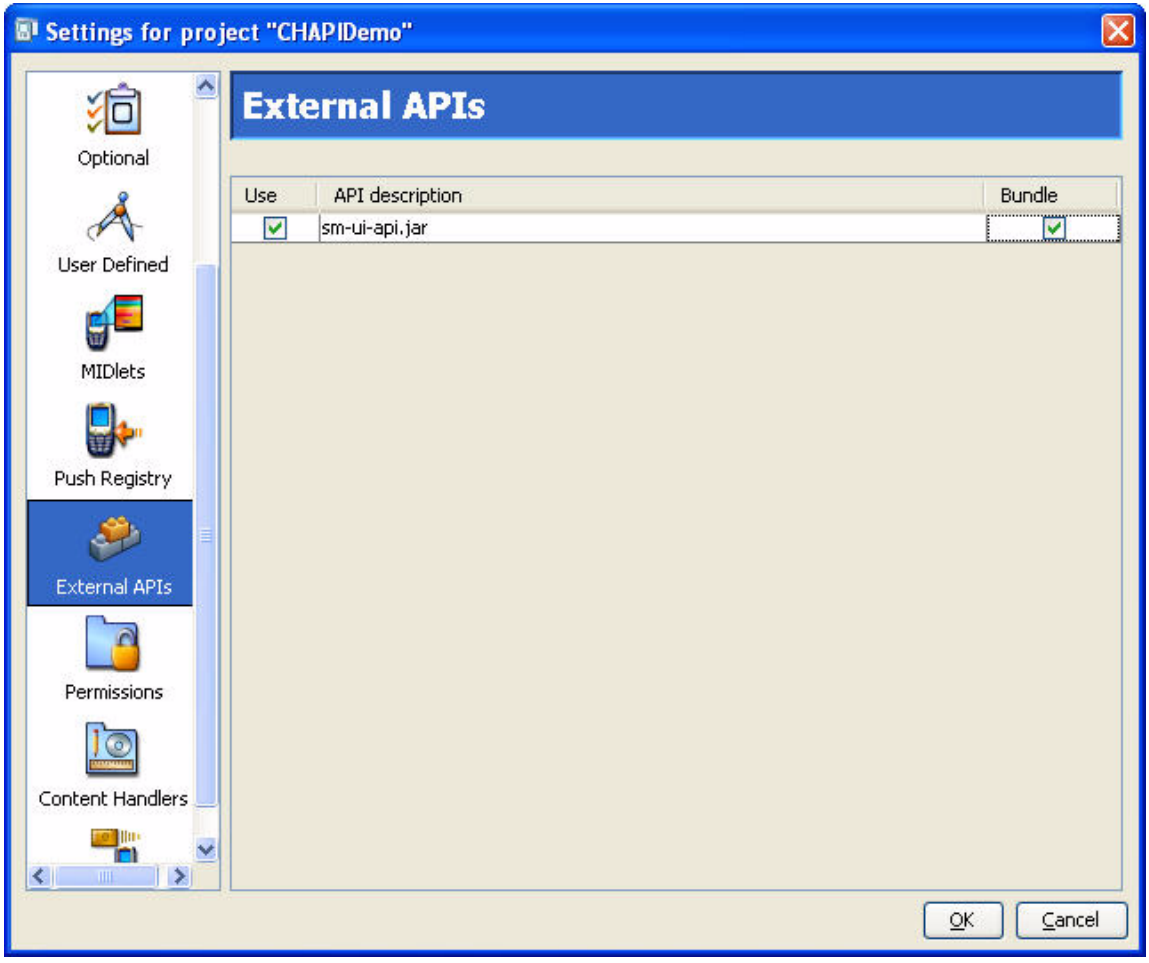
The Sun Java™ Wireless Toolkit for CLDC supplies two methods for incorporating third-party libraries. The External APIs pane in the project settings makes it easy to include or exclude libraries in a project. In addition, you can place libraries in specific locations to make them available to one or all projects.

3.7.1 Using External APIs

To specify which APIs should be included, click Settings and click the External APIs icon. A list of available external APIs appears (the list might be empty).

To add the API to the classpath at build time, check the box in the Use column (see FIGURE 3-8). If you want the API bundled into your application, also check the Bundle box. If you know you will deploy your application on devices where your selected external APIs are not present, you need to bundle those APIs in your application.

FIGURE 3-8 Choosing External APIs



3.7.2 Third-Party Libraries for One Project

Any library files placed in your project's `lib` directory are included in the building and packaging of your project. Libraries should be JAR or Zip files of Java technology classes.

So, if you installed on Windows and your application is called `Tiny`, the class library goes in *workdir*`\apps\Tiny\lib`. When you build, run, debug, and package your project, the class files in the `lib` directory are used.

3.7.3 Third-Party Libraries for All Projects

Some devices have libraries available to all installed MIDlet suites. A manufacturer, for example, can make additional APIs available on all their devices. In this case, you want to be able to use these libraries when you build and test your application. You don't want the libraries to be included in your packaged MIDlet suite because you are installing the MIDlet suite on devices where the library is already present.

You can accomplish this by placing libraries in the *workdir*`\apps\lib` directory. Libraries in this directory are available for all projects.

3.8 Configuring the Wireless Toolkit

The toolkit includes some advanced configuration options. You can use these options by editing a copy of the `ktools.properties` file, which is found in the following location:

Windows:	*toolkit*`\wtklib\`Windows`\ktools.properties`
Linux:	*toolkit*`/wtklib/`Linux`/ktools.properties`

Copy `ktools.properties` to *workdir*`/`*os*`/wtklib` and make changes as described in the remainder of this section.

- Changing the Console Font
- Setting the Application Directory
- Setting the `javac` Encoding Property
- Working with Revision Control Systems

The effect of changes to `ktools.properies` is visible the next time the toolkit is started.

3.8.1 Changing the Console Font

You can change the font that's used in the console (and other text areas) by editing two properties. Here is one example that changes the font to Times New Roman 20 point:

```
font.JTextArea=Times New Roman
font.size.JTextArea=20
```

To restore the default font and size, remove both properties.

3.8.2 Setting the Application Directory

By default, the Sun Java™ Wireless Toolkit for CLDC stores applications in your working directory's `apps` subdirectory. You can change this by adding a line to `ktools.properties` of the following form:

`kvem.apps.dir:` *application-directory*

For Windows, any backslash (`\`) characters in the directory's path must be preceded by another backslash. Also, the directory's path cannot contain any spaces.

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

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

Linux paths can be specified as usual.

3.8.3 Setting the `javac` Encoding Property

By default, the `javac` uses the encoding set in the Java SE platform that you are running. For information on how to override the default source file encoding, see Appendix C.

3.8.4 Working with Revision Control Systems

Using the `filterRevisionControl` property, you can configure the toolkit to recognize and ignore auxiliary files created by the SCCS, RCS and CVS revision control systems.

To recognize and ignore auxiliary files, include the following line in `ktools.properties`:

```
kvem.filterRevisionControl: true
```

This prevents the toolkit from treating revision control files as source and resource files. For example, the toolkit treats a file named `src\SCCS\s.MyClass.java` as being an SCCS revision control file and not a Java technology source file.

CHAPTER 4

Using the Emulator

The Sun Java™ Wireless Toolkit for CLDC emulator simulates a MIDP device on your desktop computer. It is a convenient way to see how your application performs in a MIDP environment and gives you a tight development cycle that is entirely contained on your desktop computer.

The emulator does not represent a specific device, but it provides correct implementations of its supported APIs.

4.1 Emulator Skins

A *skin* is a thin layer on top of the emulator implementation that provides it with a certain appearance, screen characteristics, and input controls. The Sun Java™ Wireless Toolkit for CLDC comes with skins that represent different kinds of devices.

TABLE 4-1 Emulator Skins

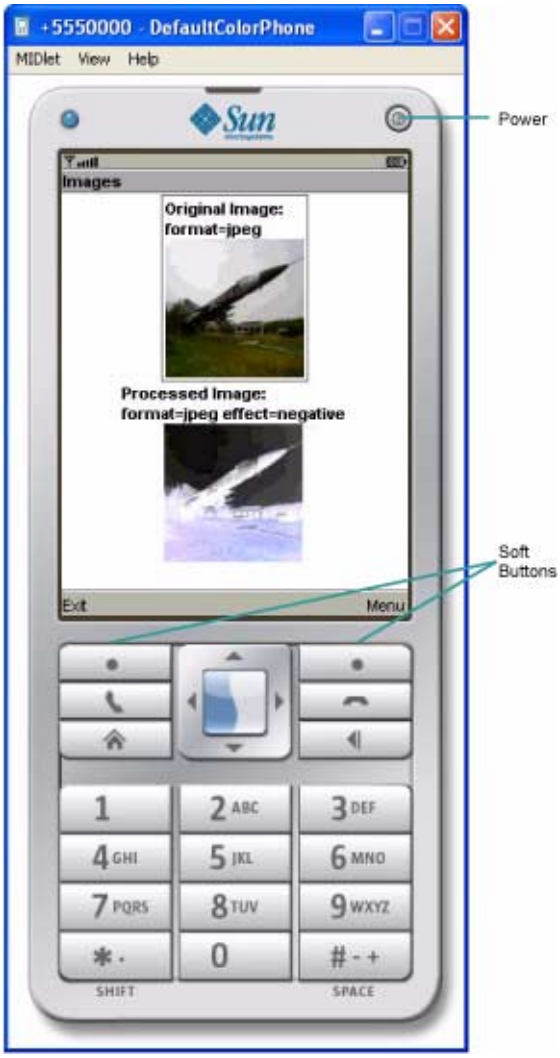
Name	Screen size	Canvas size	Colors	Input
DefaultColorPhone	240 x 320	240 x 289	4096	ITU-T
DefaultGrayPhone	180 x 208	180 x 177	4096	ITU-T
MediaControlSkin	180 x 208	180 x 177	4096	ITU-T
QwertyDevice	636 x 235	540 x 204	4096	Qwerty

You can create your own emulator skins if you wish. See the *Basic Customization Guide* for details.

4.2 Emulator Controls

The emulator looks and acts like a mobile phone inside a standard desktop window. This section describes how to control the emulator. The description and figures are based on the `DefaultColorPhone` skin, but all the skins operate in a similar way.

FIGURE 4-1 The `DefaultColorPhone` Emulator skin



You can use the mouse to click the buttons to press them. Most buttons also have keyboard shortcuts, which are generally easier to use. Keyboard numbers 0 through 9 correspond to the emulator's 0 through 9 buttons. Some less obvious keyboard shortcuts are in the following table.

TABLE 4-2 Keyboard Shortcuts

Emulator Button	Keyboard Key
Left soft button	F1
Right soft button	F2
Power button	Esc
SELECT	Enter

Entering text works much as it does on many real devices. Press a number key multiple times to get the letter you want. For example, press the 5 key twice for the letter K. When you are entering text, the asterisk key (*) switches between upper case, lower case, numbers, and symbols. The indicator at the top of the screen shows your current mode. The pound key (#) enters a space.

Alternatively, you can just type on your keyboard to enter text. Although this is convenient for entering text, you must remember that it is a convenience your users will most likely be lacking.

Another convenience is the capability to copy and paste information in text areas. You can paste text from the clipboard into a TextBox or TextField by pressing Control-v. To copy the contents of a TextBox or TextField, press Control-c. The entire contents of the text field will be placed on the clipboard.

4.3 Setting Emulator Preferences

You can adjust the emulator settings to more closely resemble a specific device or to test your application under different resource conditions.

4.3.1 Network Proxies

The emulator uses your desktop network connection. For example, if the emulator runs a MIDlet that makes an HTTP connection, the emulator attempts to make the HTTP connection using the desktop's network setup.

If your development computer is behind a firewall, you might use a proxy server to make HTTP connections. If you're not sure, try examining your browser's settings to see if it uses proxy servers.

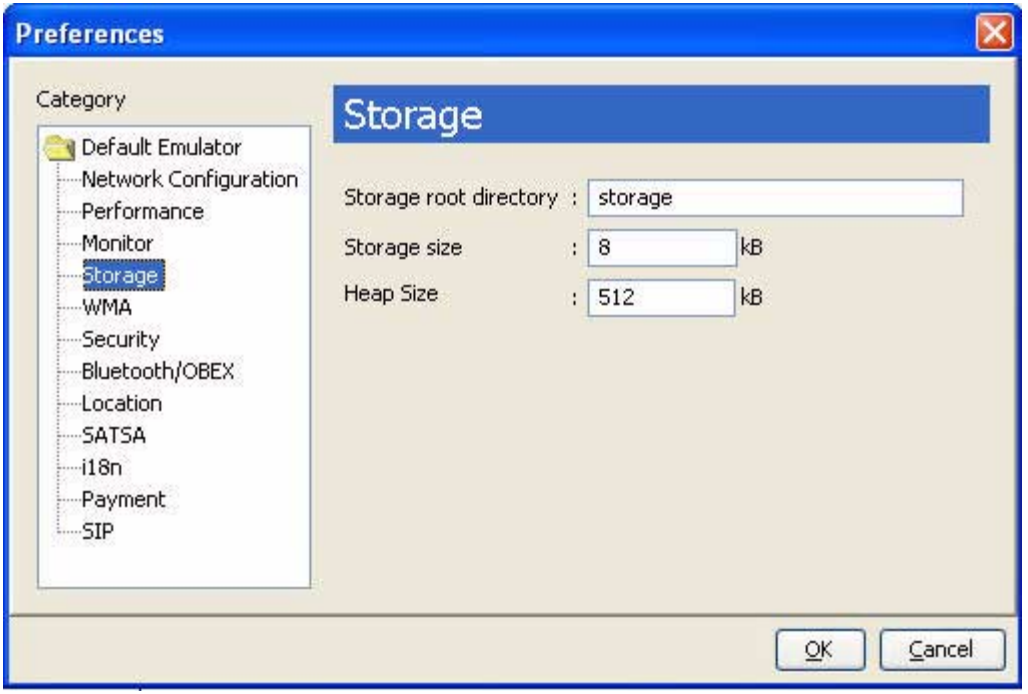
If you are using proxy servers, you need to configure the emulator to use the same proxy servers. To do this, choose Edit > Preferences. On the Network Configuration pane, fill in the names and port numbers for the proxy server you wish to use. You can also select which version of HTTP you wish to use.

If your proxy servers use HTTP Basic authentication (see RFC 2617), check Authentication and fill in the user name and password.

4.3.2 Storage Sizes

You can set or modify the amount of persistent storage assigned to each emulator, or you can change the heap size (the amount of storage allowed for your application's objects). Choose Edit > Preferences and select the Storage item.

FIGURE 4-2 Storage Preferences



4.3.2.1 Persistent Storage

The emulator has persistent storage, which by default is placed below the `appdb` subdirectory in a phone skin directory. These files have a `.db` extension.

For example, on Windows the persistent storage for the `DefaultColorPhone` emulator skin is stored in *workdir*`\appdb\DefaultColorPhone\manager_storage_settings.db`.

If multiple instances of the same emulator skin run simultaneously, the toolkit generates unique file paths for each one. For example, on Windows instances of `DefaultColorPhone` might have a file path name of *workdir*`\appdb\temp.DefaultColorPhone1`, *workdir*`\appdb\temp.DefaultColorPhone2`, and so forth.

Note – The file *workdir*`\appdb\DefaultColorPhone\in.use` keeps track of the number of storage roots marked as in use. If the emulator crashes, you need to delete the `in.use` file.

The toolkit enables you to choose a different location for the storage files, and you can limit the size of the storage. This is useful if you wish to test your application's behavior when a small amount of persistent storage is available.

To adjust the persistent storage settings, choose Edit > Preferences and click Storage in the left pane. In the Storage root directory field, enter the name of the directory you want to use for persistent storage. You can only enter a relative path, and the directory you specify is created in the `appdb` subdirectory.

By default you are allowed one megabyte (1024 Kbytes) of persistent storage. You can enter a limit in kilobytes. Bear in mind that the storage implementation has some overhead in addition to the space your application uses. For example, if you enter 8 kilobytes for the persistent storage size, 8192 bytes is available for both your application data and the storage overhead.

To erase the emulator's persistent storage, choose File > Utilities. Click the Clean Database button to wipe the persistent storage clean. Clean Database does not affect installed applications.

4.3.2.2 Heap Size

The *heap* is memory where your application's objects are stored. To change the heap size, choose Edit > Preferences and select the Storage item (see FIGURE 4-2). By default the heap size is one megabyte. You can set the maximum heap size to more closely simulate the conditions on a real device. Fill in the maximum heap size in kilobytes in the Heap Size field.

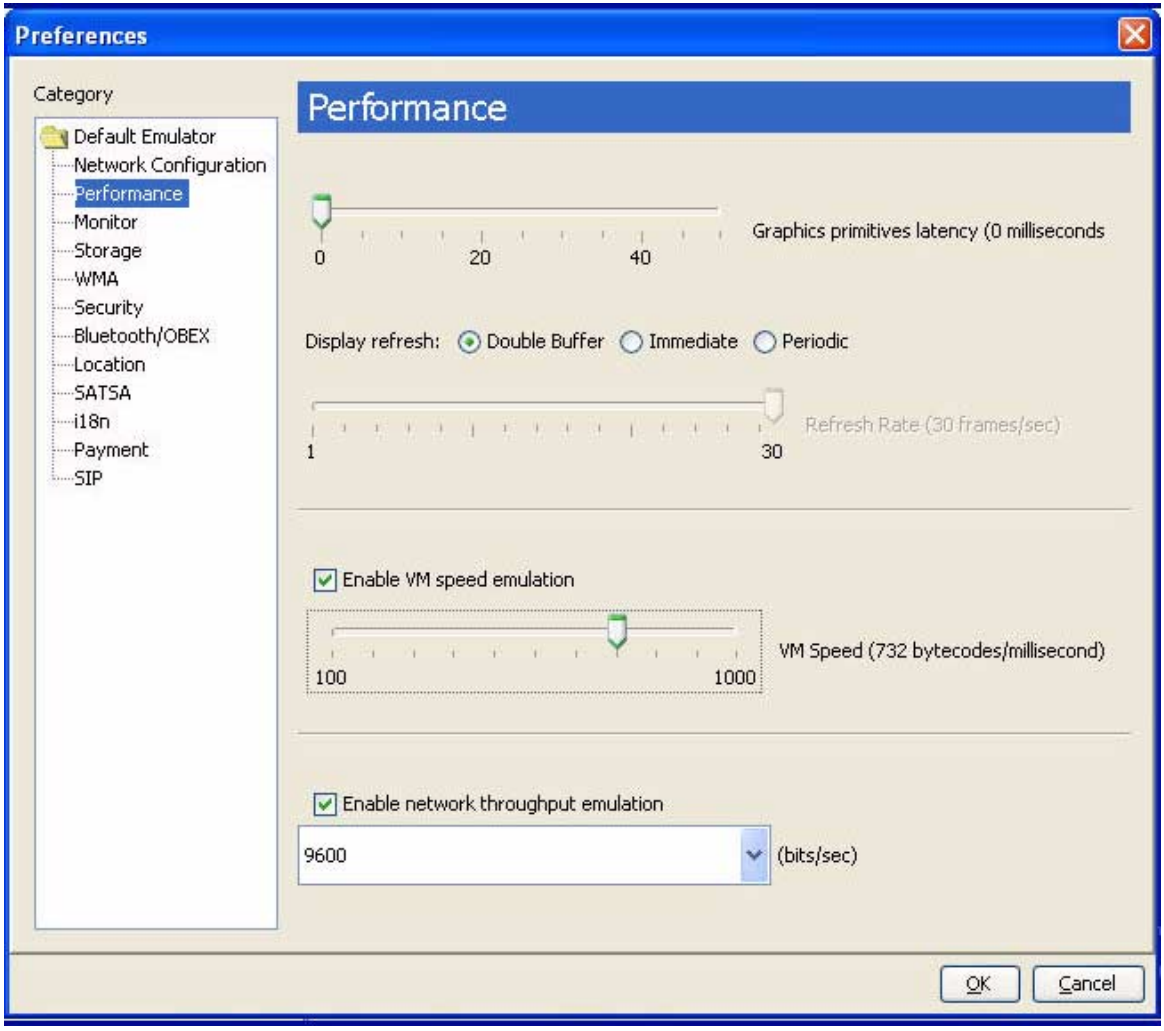
4.3.3 Adjusting Emulator Performance

The emulator uses many of the resources of your desktop computer, including its display and network connection. Compared to the desktop-based emulator, a real MIDP device usually has a slower processor, less memory, and a slower network connection, and might have a different type of display.

The Sun Java™ Wireless Toolkit for CLDC enables you to simulate the constrained environment of a real device. Although the emulator does not represent a real device, adjusting the performance settings gives you useful information about how your application performs under varying runtime conditions.

Choose Edit > Preferences and click Performance in the left pane. See FIGURE 4-3.

FIGURE 4-3 Adjusting the Emulator's Performance



Adjust the Graphics primitives latency to have an effect on the amount of time that elapses between your application's calls to drawing methods in the `Graphics` class and when the drawing actually takes place.

To change the screen characteristics, choose one of the Display refresh types. If you choose a Periodic type, you also need to specify the Refresh Rate.

To simulate the slower speed of a real device, check Enable VM speed emulation and choose the speed you want.

To adjust the simulated network speed, check Enable network throughput emulation, and choose a speed in bits/sec.

4.4 Pausing and Resuming

MIDlets have a life cycle that is defined by the MIDP specification. MIDlets can be started and stopped by the device. Furthermore, external events like incoming phone calls can cause the device to pause a MIDlet.

The emulator provides a simple mechanism to pause and resume running MIDlets. This is very helpful for testing your application's behavior when it is paused.

When the emulator is running, choose MIDlet > Pause from the emulator window's menu. The running MIDlet is paused and the screen displays an "Incoming Call..." message.

To resume operation, choose MIDlet > Resume from the menu.

4.5 Running the Emulator Solo

During development, you usually run the emulator directly from the toolkit by pressing the Run button or selecting Project > Run via OTA. For testing or demonstrations, you might want to run the emulator by itself. Several different approaches are described in this section. The program group that the Sun Java™ Wireless Toolkit for CLDC installer creates includes several options for running the emulator by itself.

- To run an application directly, which is analogous to pressing the Run button, choose the Run MIDP Application... item. The toolkit prompts you to locate a MIDlet descriptor file on your local disk. Note that the corresponding MIDlet suite JAR must also be present.
- To run the emulator's Application Management Software (AMS), choose the OTA Provisioning item, which is roughly analogous to Run via OTA feature in the user interface. The emulator pops up with the AMS welcome screen, and you can install applications by typing in a URL.
- To change the emulator's preferences, choose the Preferences item from the toolkit program group. This pulls up the same preferences window as choosing Edit > Preferences... in the user interface.
- The Sun Java™ Wireless Toolkit for CLDC utilities are also accessible without running the user interface. Just choose the Utilities item.
- Finally, you can change which emulator skin is used by default. Choose the Default Device Selection item, and choose one of the available emulator skins. The next time you launch the emulator the selected skin is used.

You can also run the emulator from a command prompt. See Appendix B for more information.

4.6 Using Third-Party Emulators

Third-party companies, like device manufacturers and wireless carriers, sometimes create device emulators that are compatible with the toolkit. You can gain experience running your application on a wider variety of implementations by installing additional emulators into the toolkit. The procedure is usually to unpack or install the third party emulator, then copy its directory into *workdir*`\wtklib\devices`. The next time you run the toolkit, the emulator is available.

A partial listing of some of the currently available emulators is available here:

`http://developers.sun.com/techtopics/mobility/midp/articles/emulators/`

CHAPTER 5

Monitoring Applications

The Sun Java™ Wireless Toolkit for CLDC provides several tools to monitor the behavior of your applications. These tools are helpful in debugging and optimizing your code:

- The *profiler* lists the frequency of use and execution time for every method in your application.
- The *memory monitor* shows the usage of memory while your application runs.
- The *network monitor* shows network data transmitted and received by your application. It supports many network protocols including HTTP, HTTPS, SMS, and CBS.
- *Tracing* outputs low-level information to the toolkit console.

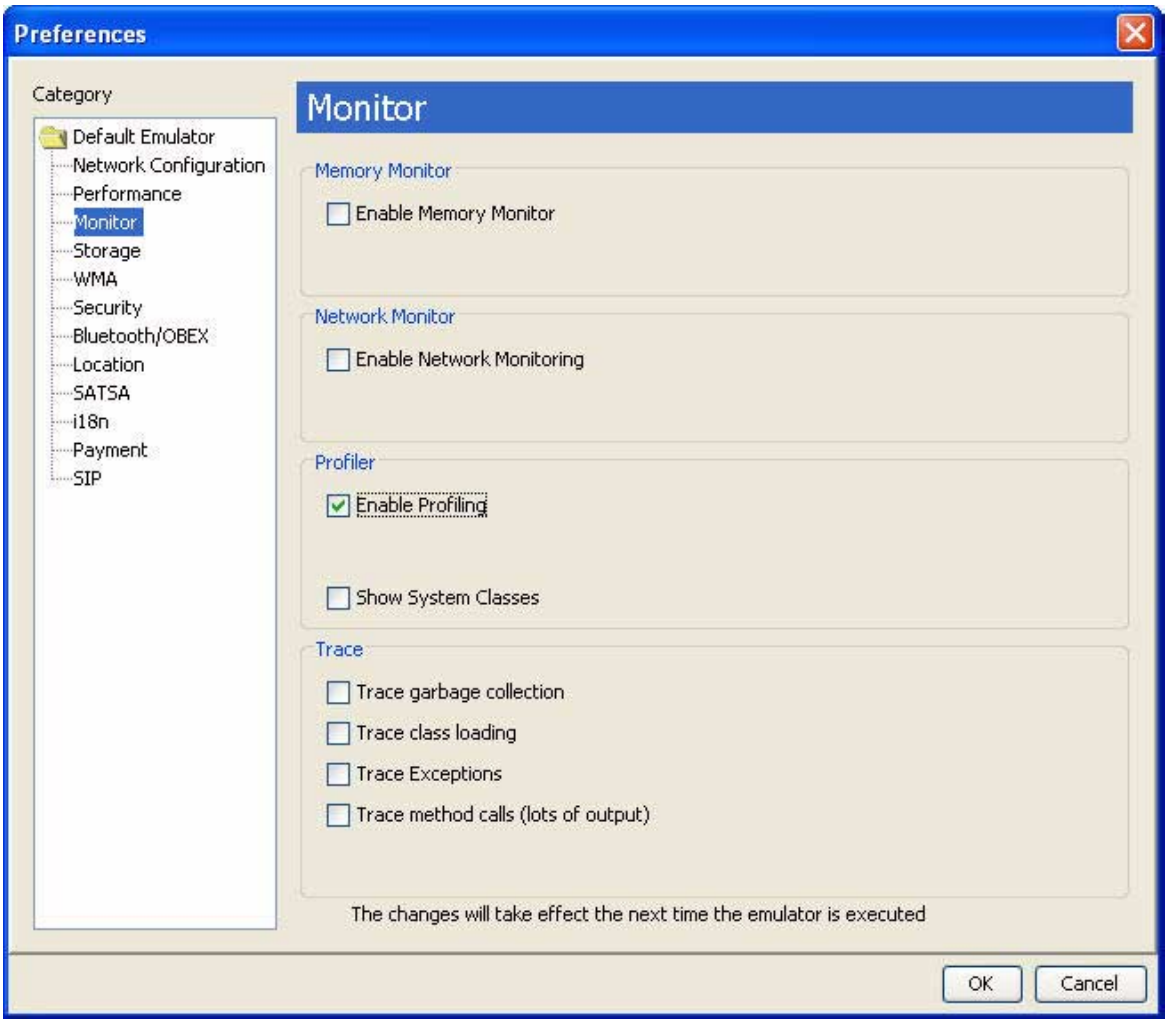
Note – Monitoring features might slow the execution of your application.

5.1 Using the Profiler

The profiler keeps track of every method in your application. For a particular run, it figures out how much time was spent in each method and how many times each method was called. After you finish running your application and shut down the emulator, the profiler pops up, allowing you to browse through the method call information.

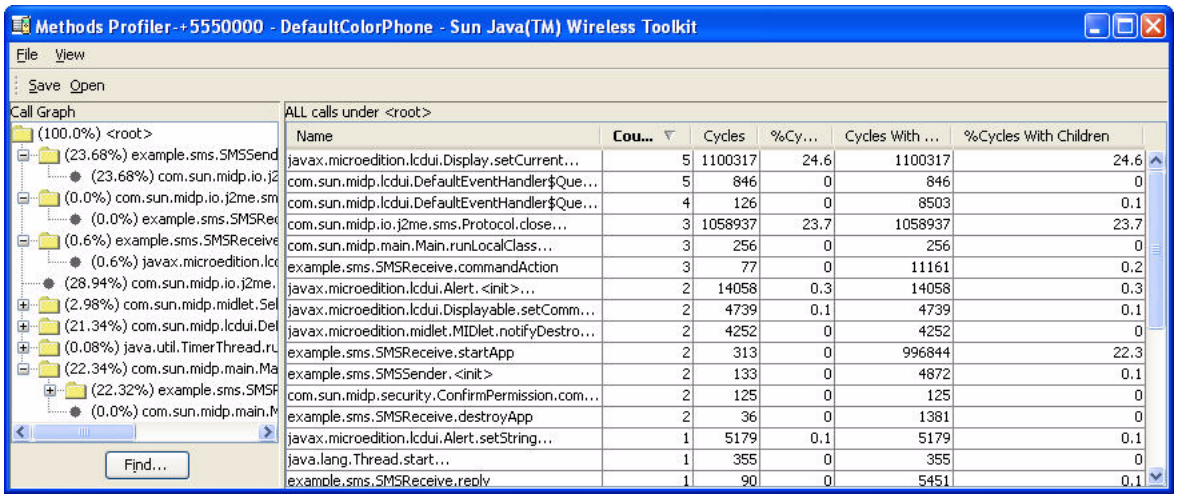
To turn on the profiler, choose Edit > Preferences. See FIGURE 5-1. Click Monitor in the left pane. In the right pane, check Enable Profiling. If you are interested in seeing profiling information for all the system implementation methods, check Show System Classes. Otherwise, the profiler shows only system methods that contain calls to your application methods. Click OK.

FIGURE 5-1 Turning on the Profiler



Now click the Run button to start your application. Interact with your application as you normally would. When you are finished, shut down the emulator. The profiler pops up with information about all the method calls in your application.

FIGURE 5-2 Method Profiler



The profiler displays two types of information:

- Method relationships are shown in a hierarchical list, the Call Graph.
- The right side of the profiler shows the execution time and number of calls for each method and its descendants.

Note – The profiling values obtained from the emulator do not reflect actual values on a real device.

5.1.1 Call Graph

The call graph shows a hierarchy of method calls. Methods that call other methods are shown as folders. Double-click a method to open it and see the methods it calls. Methods that do not call any other method are shown as gray circles.

You can search for a particular class or method name. Click Find and fill in a name. The search is performed from the current selection in the call graph to the end. If you want to search the entire call graph, check Wrap before you click the Find button.

As you click different nodes in the call graph, the right side of the profiler shows details about the methods for that node.

5.1.2 Execution Time and Number of Calls

The right side of the profiler window displays detailed information about methods. You can see the method name, the number of times it was called, and the amount of time that the emulator spent in the method. The execution time is described in four distinct ways:

- Cycles shows the amount of processor time spent in the method itself.
- %Cycles is the percentage of the total execution time that is spent in the method itself.
- Cycles with Children is the amount of time spent in the method *and* its called methods.
- %Cycles with Children shows the time spent in the method and its called methods as compared to the total execution time.

Click any column to sort by that column. Click a second time to switch the sort between ascending and descending.

The right pane shows the methods contained in the currently selected node in the call graph. If you want to see every method, click on the <root> node in the call graph.

5.1.3 Saving and Loading Profiler Information

To save your profiler session, click the Save button in the profiler window. Choose a file name.

To load a profiler session, choose File > Utilities. Click Profiler and press Launch. When you select a file, the profiler window appears with all the session information.

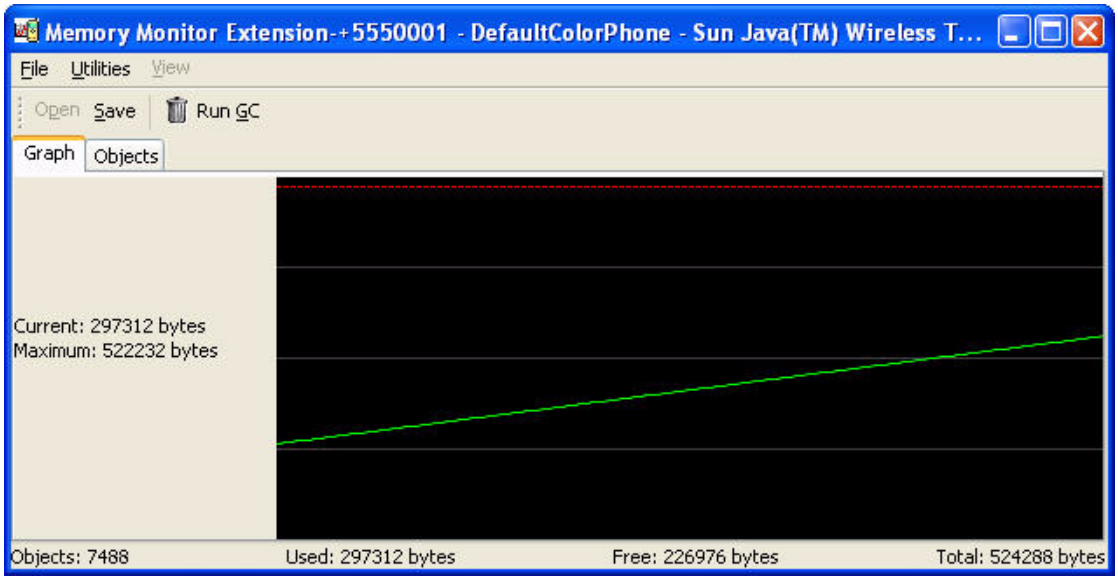
5.2 Using the Memory Monitor

Memory is scarce on many MIDP devices. The Sun Java™ Wireless Toolkit for CLDC includes a memory monitor that makes it easy to examine the memory usage of your application. You can see the total memory used by your application and a detailed listing of the memory usage per object.

To turn on the memory monitor, choose Edit > Preferences. Click on Monitor in the left pane. Check Enable Memory Monitor.

Next time you run the emulator, the memory monitor window appears, displaying a graph of your application's memory usage over time. The memory monitor slows your application startup because every object created is recorded.

FIGURE 5-3 Memory Monitor Graph



The memory monitor graph shows the following information:

- **Current -** Current amount of memory used by the application.
- **Maximum -** Maximum amount of memory used since program execution began, shown in the graph by a broken red line.
- **Objects -** Number of objects in the heap.
- **Used -** Amount of memory used.
- **Free -** Amount of unused memory available.
- **Total -** Total amount of memory available at startup.

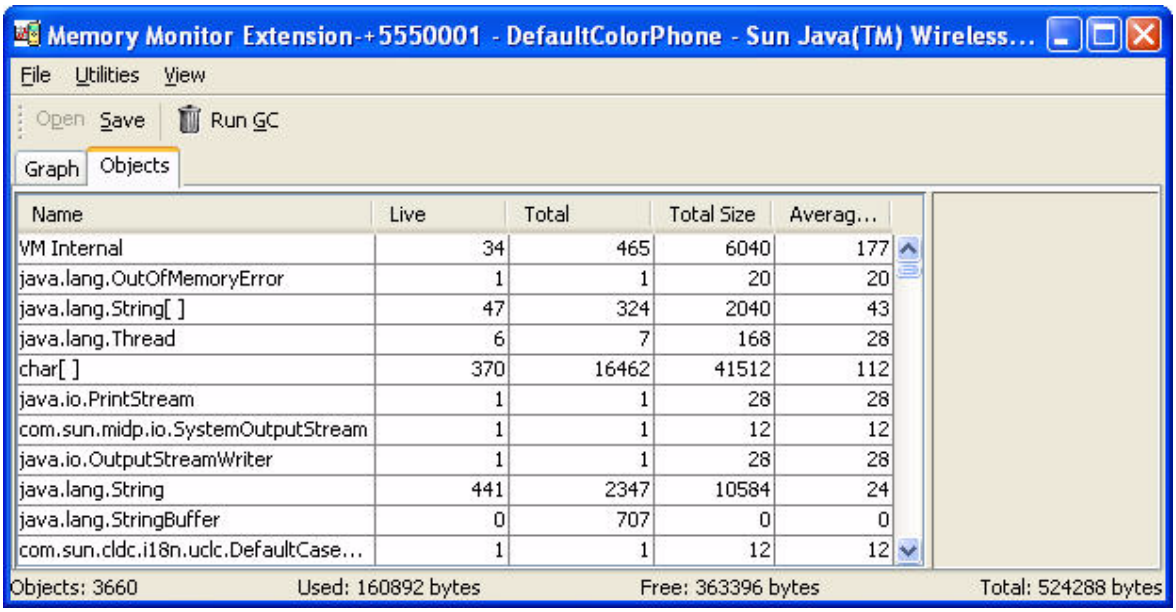
Remember, to modify the heap size select Edit > Preferences and choose the Storage tab. See Chapter 3 for details.

To request the system to perform a garbage collection, click Run GC.

Note – The memory usage you observe with the emulator is not exactly the same as memory usage on a real device. Remember, the emulator does not represent a real device. It is just one possible implementation of its supported APIs.

To see details about the objects in your application, click the Objects tab in the memory monitor window.

FIGURE 5-4 Memory Monitor Objects Display



A table with the following columns appears:

- **Name -** Class name of the objects.
- **Live -** Number of instances. Some of these might be eligible for garbage collection.
- **Total -** Total number of objects that have been allocated since the application began.
- **Total Size -** Total amount of memory used by the objects.
- **Average Size -** Average object size, calculated by dividing the live instances into the total size.

Click any column header to sort on that column.

You can search for a specific class name by choosing View > Find... from the memory monitor window menu.

5.2.1 Saving and Loading Memory Monitor Information

To save your memory monitor session, click the Save button. Choose a file name.

To load a memory monitor session, choose File > Utilities. Click Memory Monitor and press Launch. When you select a file, the memory monitor window appears with all the session information.

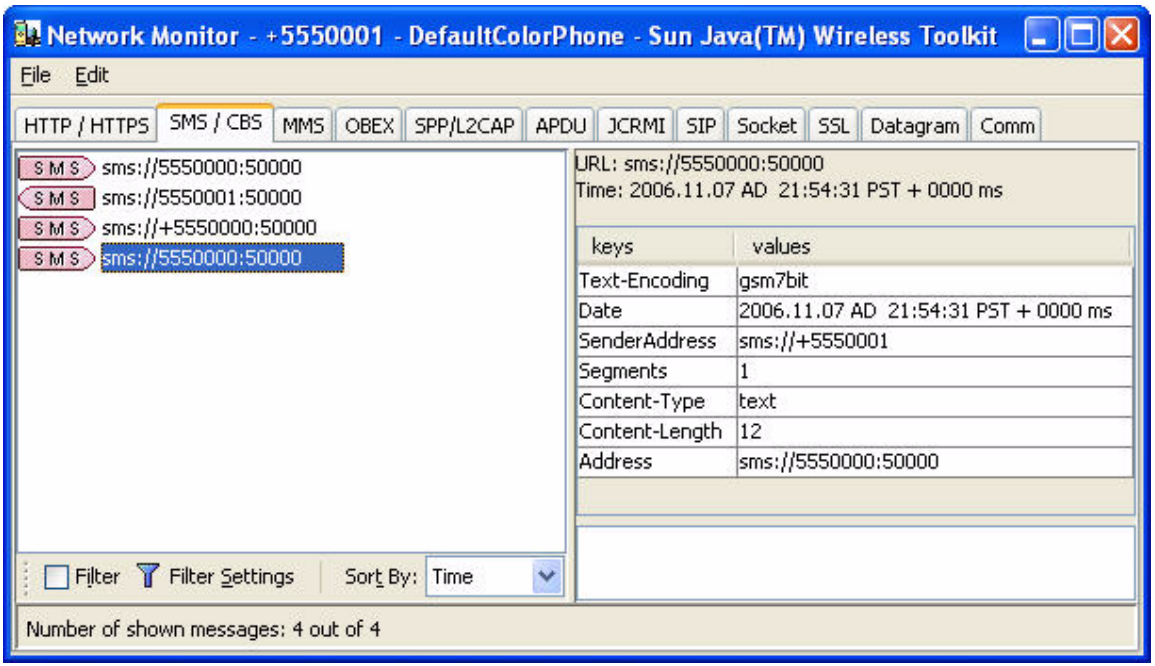
5.3 Using the Network Monitor

MIDP applications, at a minimum, are capable of HTTP network connections, but many other types of network connections are also possible. The network monitor provides a convenient way to see the information your application is sending and receiving on the network. This is helpful if you are debugging network interactions or looking for ways to optimize network traffic.

To turn on the network monitor, choose Edit > Preferences. Click Monitor in the left pane. Check Enable Network Monitoring.

Next time you run the emulator, the network monitor window appears.

FIGURE 5-5 Network Monitor



When your application makes any type of network connection, information about the connection is captured and displayed. The figure shows HTTP requests and responses.

The display on the left side shows a hierarchy of messages and message pieces. Click a message or a portion of a message to see details in the right side of the network monitor. Double-click messages or message portions to expand or collapse them.

Message bodies are shown as raw hexadecimal values and the equivalent text.

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.

5.3.1 Filtering Messages

Filters are useful for examining some subset of the total network traffic. Filter settings are specific to the network protocol used.

Press the Filter Settings button to use the filter. Change the filter settings to suit your needs.

TABLE 5-1 Network Monitor Filter Settings

Network Protocol	Filter Settings
HTTP/HTTPS	Enter text to match the various parts of HTTP messages: URL, status line, headers, or body. For example, entering `slashdot` in the URL field filters to show only messages whose URL contain `slashdot`.
SMS/CBS	You can specify a protocol, message type, and direction to match. Furthermore, you can enter text to match in the sender, receiver, and message content.
MMS	Enter text to match the direction, sender, receiver, and copied (cc) and blind copied (bcc) receivers. In addition, you can filter on the subject, content ID, content location, MIME type, and encoding.
OBEX, SPP/L2CAP	You can filter using the URL or header content.
APDU, JCRMI	Filter on the URL or the message content.
SIP	None available
Socket, SSL, Datagram, Comm	Enter text to match in either the connection string (URL) or content.

When you are done entering filter settings, press OK to return to the network monitor. The Filter checkbox is checked, indicating that a filter is in use. To disable the filter and see all messages, uncheck the checkbox.

5.3.2 Sorting Messages

To arrange the message tree in a particular order, click on the Sort By combo box and choose a 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.
- Sorting parameters are dependent on the message protocol you choose. For instance, sorting by time is not relevant for socket messages.

5.3.3 Saving and Loading Network Monitor Information

To save your network monitor session, choose File > Save or File > Save As from the network monitor window menu. Choose a file name.

To load a network monitor session, choose File > Utilities. Select Network Monitor from the list and press Launch. When you select a file, the network monitor window appears with all the session information.

5.3.4 Clearing the Message Tree

To remove all messages from the network monitor, choose Edit > Clear from the network monitor menu.

CHAPTER 6

Security and MIDlet Signing

MIDP 2.0 (JSR 118) includes a comprehensive security model based on protection domains. MIDlet suites are installed into a protection domain that determines access to protected functions. The MIDP 2.0 specification also includes a recommended practice for using public key cryptography to verify and authenticate MIDlet suites.

For definitive information, consult the MIDP 2.0 specification. For an overview of MIDlet signing using the Sun Java™ Wireless Toolkit for CLDC, read the article *Understanding MIDP 2.0's Security Architecture*, which is available at http://developers.sun.com/techtopics/mobility/midp/articles/permissions/

If you need more background on public key cryptography, try the article *MIDP Application Security 1: Design Concerns and Cryptography*, which is available at http://developers.sun.com/techtopics/mobility/midp/articles/security1/

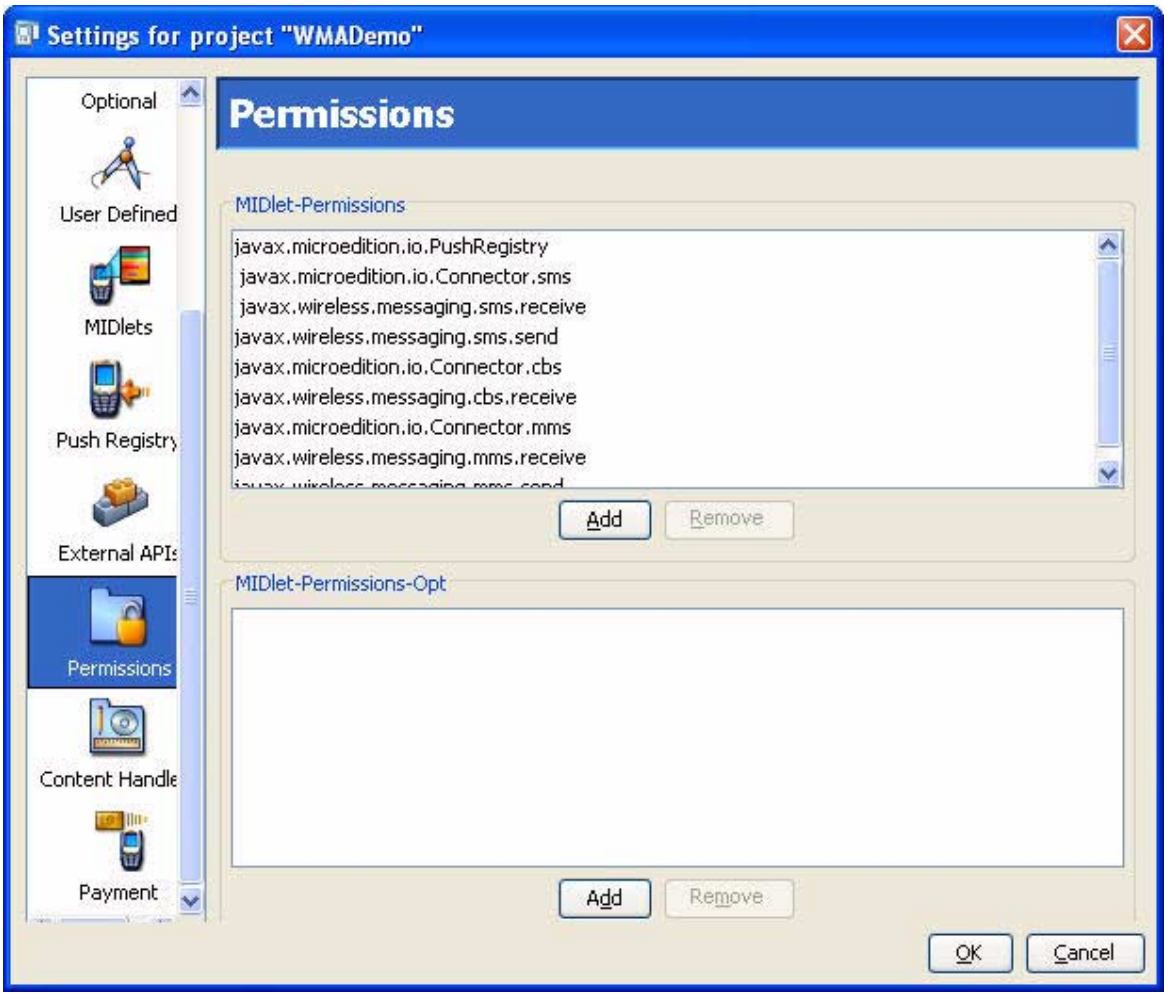
This chapter describes support for protection domains, permissions, and MIDlet signing in the Sun Java™ Wireless Toolkit for CLDC.

6.1 Permissions

MIDlets must have permission to perform sensitive operations, such as connecting to the network. Permissions have specific names, and MIDlet suites can indicate their need for certain kinds of permissions through attributes in the MIDlet suite descriptor.

You can add these permission attributes to a project by clicking the Settings button. Select the Permissions icon. The MIDlet-Permissions box shows permissions which the MIDlet must possess, while the MIDlet-Permissions-Opt box contains permissions that are optional.

FIGURE 6-1 MIDlet Suite Permissions



To add a permission to either box, click Add and choose the permission you want to add. To remove a permission, highlight it and click Remove.

6.2 Selecting the Security Policy

The Sun Java™ Wireless Toolkit for CLDC supports the security policies defined by both JSR 185 (Java Technology for the Wireless Industry) and JSR 248 (Mobile Service Architecture or MSA). The protection domains are further described in Section 6.2.1, "MSA Protection Domains" on page 6-3 and Section 6.2.2, "Java for the Wireless Toolkit Industry Protection Domains" on page 6-4.

To choose the security policy you want the emulator to use, select Edit > Preferences and select Security in the Category list. From the Security Policy combo box, choose either MSA or JTWI. Select one of the available security policies.

When you use Run via OTA. your packaged MIDlet suite is installed directly into the emulator and it is placed in a protection domain at installation time. The emulator uses public key cryptography to determine the protection domain of installed MIDlet suites.

If the MIDlet suite is not signed, it is placed in the default protection domain. The default is different for MSA and JTWI. See Sections 6.2.1 and 6.2.2. If the MIDlet is signed, it is placed in the protection domain that is associated with the root certificate of the signing key's certificate chain.

For example, suppose Respectable Software, a hypothetical company, wants to distribute a cryptographically signed MIDlet suite. Respectable Software buys a signing key pair from Super-Trustee, a hypothetical certificate authority. Using the signing key, Respectable Software signs the MIDlet suite and distributes their certificate with the MIDlet suite. When the MIDlet suite is installed on the emulator or on a device, the implementation verifies Respectable's certificate using its own copy of Super-Trustee's root certificate. Then it uses Respectable's certificate to verify the signature on the MIDlet suite. Assuming everything checks out, the device or emulator installs the MIDlet suite into the protection domain that is associated with Super-Trustee's root certificate, most likely `identified_third_party`.

The toolkit provides tools to sign MIDlet suites, manage keys, and manage root certificates.

6.2.1 MSA Protection Domains

The toolkit supports five protection domains for MSA:

- `unidentified_third_party` - Provides a high level of security for applications whose origins and authenticity cannot be determined. The user is prompted frequently when the application attempts a sensitive operation

- `identified_third_party` - Intended for MIDlets whose origins were determined using cryptographic certificates. Permissions are not granted automatically, but the user is prompted less often than for the `unidentified_third_party` domain.
- `manufacturer` - Intended for MIDlet suites whose credentials originate from the manufacturer's root certificate.
- `minimum` - All permissions are denied to MIDlets in this domain.
- `maximum` - All permissions are granted to MIDlets in this domain.

When you press the Run button to run your application in the emulator, your code runs in the `unidentified_third_party` protection domain by default.

6.2.2 Java for the Wireless Toolkit Industry Protection Domains

The Sun Java Wireless Toolkit includes four protection domains:

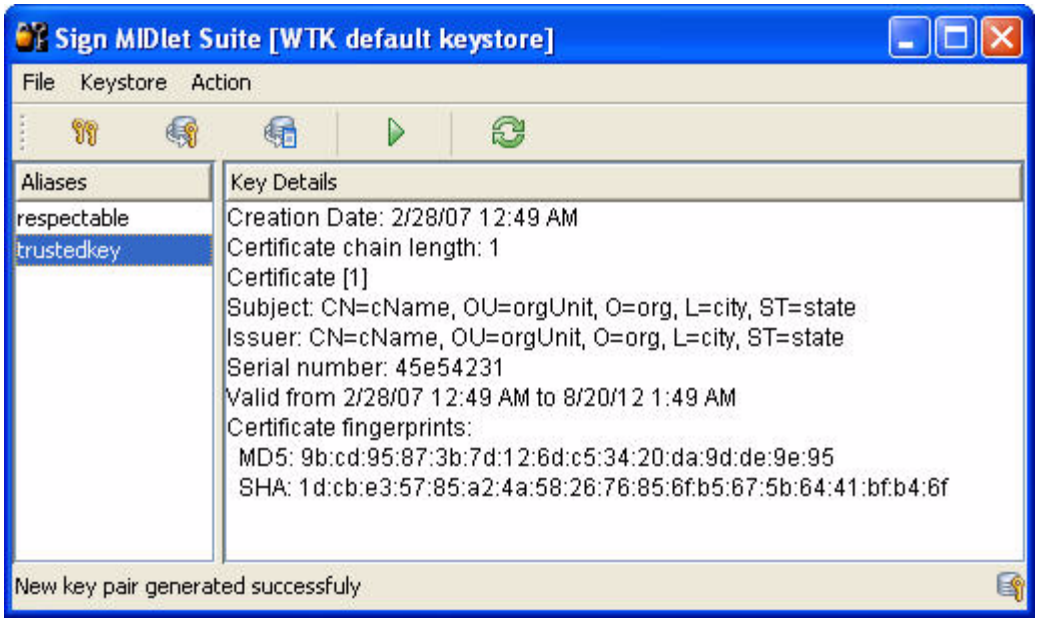
- `untrusted` - Provides a high level of security for applications whose origins and authenticity cannot be determined. The user is prompted frequently when the application attempts a sensitive operation.
- `trusted` - All permissions are granted to MIDlets in this domain.
- `minimum` - All permissions are denied to MIDlets in this domain.
- `maximum` - All permissions are granted to MIDlets in this domain (equivalent to `trusted`.).

When you press the Run button to run your application in the emulator, your code runs in the `untrusted` protection domain by default.

6.3 Signing a MIDlet Suite

To sign a MIDlet suite, you must package it first (select Project > Package). Then choose Project > Sign. The signing window appears (FIGURE 6-2).

FIGURE 6-2 MIDlet Suite Signing Window



Select the key you want to use in the Alias List and click the Sign MIDlet Suite button (the green triangle).

6.4 Managing Keys

The MIDlet signing window can also be used to manage keys. For test purposes, you can create a key pair to sign a MIDLet. To deploy on a device, you must obtain a signing key pair from a certificate authority recognized by the device. You can also import keys from an existing Java SE platform keystore.

6.4.1 Creating a New Key Pair

To create an entirely new key pair, click New Key Pair. The toolkit prompts you for a key alias and information that will be associated with the key pair.

FIGURE 6-3 Creating a New Key Pair

After you click Create, the toolkit prompts you to choose a protection domain. The connection between the key pair you just created and a protection domain might seem oblique, but it makes perfect sense:

- The toolkit creates a self-signed root certificate using the key pair you just created.
- The root certificate is added to the emulator's list of root certificates.
- The toolkit needs to associate the root certificate with a protection domain.

Now imagine what happens when you install a MIDlet suite signed with your new key:

- The implementation examines the certificate chain in the MIDlet suite descriptor. In this case the certificate chain is a single certificate, the self-signed root.
- The implementation tries to find the root of the certificate chain in its internal list. This succeeds because the root certificate was added when you create the key pair.
- The implementation considers the certificate valid and uses it to verify the signature on the MIDlet suite.
- The MIDlet suite is installed into whatever protection domain you picked.

6.4.2 Getting Real Keys

The ability to create a key pair and sign a MIDlet within the Sun Java™ Wireless Toolkit for CLDC environment is for testing purposes only. When you run your application on an actual device, you must obtain a signing key pair from a certificate authority recognized by the device.

The procedure for signing MIDlet suites with real keys works this way:

1. **Generate a new key pair.**

   In the Sun Java™ Wireless Toolkit for CLDC, you can do this by pressing New Key Pair in the MIDlet signing window, as described above.

2. **Generate a Certificate Signing Request (CSR).**

   a. **Press Generate CSR in the signing window.**

   b. **To change the location of the CSR file, enter a new path or press Browse and choose a new file location.**

   c. **Press Create to write the CSR file.**

      After the CSR is written, a message that indicates success appears.

3. **Send the CSR to a certificate authority (CA).**

   The CA needs more information from you to verify your identity. You must also pay the CA for the certificate they generate for you.

   Once the CA verifies your identity and taken your money, it sends a certificate that certifies your public key.

4. **Import the certificate into the Sun Java™ Wireless Toolkit for CLDC by pressing Import Certificate... in the MIDlet signing window.**

You can now use your own private key to sign MIDlet suites. The Sun Java™ Wireless Toolkit for CLDC takes care of the details of placing the signature and your certificate into the MIDlet suite.

6.4.3 Importing an Existing Key Pair

You might have keys in a Java SE platform keystore that you would like to use for MIDlet signing. In this case, you need to import your signing keys to the Sun Java™ Wireless Toolkit for CLDC so that you can sign your MIDlet suite. To do this from the MIDlet signing window, click Import Key Pair. Select a file that contains a Java SE platform keystore. You are prompted to select the alias of the key pair you want to import, then you just supply the alias you want to identify the key pair once it is imported to your keystore. Finally, you must select a protection domain for the key pair's root certificate.

6.4.4 Removing a Key Pair

To remove a key pair from the MIDlet signing window, select its alias and choose Action > Delete Selection.

6.5 Managing Certificates

This section describes how to manage the emulator's list of root certificates using the Sun Java™ Wireless Toolkit for CLDC.

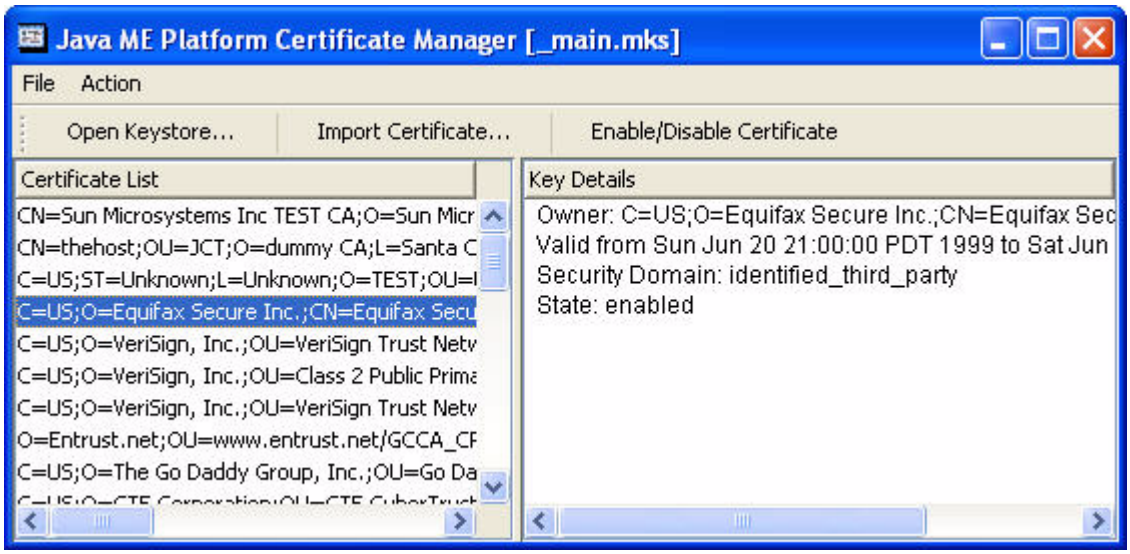
Real devices have similar lists of root certificates, although they cannot usually be modified by the user. When you want to deploy your application on a real device, you must to use signing keys issued by a certificate authority whose root certificate is present on the device. Otherwise, the device is unable to verify your application.

While you're developing your application, the toolkit's certificate management utility provides a convenient way to manipulate the emulator's list of root certificates for testing purposes.

Choose File > Utilities. Select Manage Certificates and press Launch to open up the certificate manager window. The micro keystore, `_main.mks` is displayed. This keystore resides in the `appdb` directory.

The `appdb` directory also contains `keystore.ks` and `serverkeystore.ks`. The Java ME Platform Certificate Manager cannot open `*.ks` files, but you can import certificates from these keystores as described in Section 6.5.2, "Importing Certificates" on page 6-9.

FIGURE 6-4 Certificate Manager



Each certificate is shown as a single line in the left part of the window, the Certificate List. When you click a certificate, its details are shown in the right part of the window along with the certificate's associated protection domain.

6.5.1 Enabling and Disabling Certificates

Certificates can be enabled or disabled. This is handy if you want to make certificates temporarily unavailable without removing them from the keystore. To enable or disable a certificate, select it in the list and press Enable/Disable Certificate. The toolkit asks you to confirm the action. Choose Yes to proceed.

6.5.2 Importing Certificates

You can import certificates either from certificate files or from Java SE platform keystore files.

To import a certificate from a file, click Import Certificate in the certificate manager window. After you locate the certificate file, choose which protection domain is associated with the certificate.

To import a certificate from a Java SE platform keystore, choose Action > Import Java SE Certificate from the menu in the certificate manager window. First, choose a protection domain for the certificate. Then select the keystore file and enter the keystore password. Finally, select the alias for the certificate you wish to import.

6.5.3 Removing Certificates

To remove a certificate from the list, select the certificate and choose Action > Delete Selection.

6.6 USB Token Support

A USB token provides portable password-protected storage for public and private keys and certificates. The Java SE PKCS#11 native interface supports access to a USB token that has a PKCS#11-compliant native driver. When the driver is installed, a PKCS#11 library is included. On Windows the library is a win32 DLL.

This section provides sample instructions for installing and using a USB Token on the Windows platform.

Note – Linux is not supported because we have not fully tested a USB token with a Linux driver. USB tokens might work on Linux if a PKCS#11-compliant native driver is available.

The remainder of this section steps through the installation and setup process required for USB token support.

6.6.1 Installing USB Token Drivers

Close all applications.

1. **Go to** `http://downloads.geotrust.com/TCSPIKEY0407203016.exe`
2. **When the File Download dialog box opens, click Save.**

   Note the directory where you save the executable.
3. **Select the executable and double-click to start the Crypto Token installation.**

   Follow the installation prompts. insert the token so you can complete the installation.

4. **The Windows New Hardware Wizard launches.**

   Follow the instructions, accepting all default actions.

5. **When the Wizard finishes, answer Yes to restart your computer.**

Resetting the USB Token Passphrase

This step is valid only for new USB tokens. All USB tokens arrive with the default passphrase, PASSWORD (all upper case). You should reset this PassPhrase.

1. **Click Start on your Windows task bar. Select All Programs > GeoTrust Token, > iKey 2000 Series Software > PassPhrase Utility.**

2. **Click Update Passphrase.**

   You are prompted to input your "old passphrase" before you can input your new one.

3. **Reset the passphrase.**

   GeoTrust and Cingular recommend making your passphrase a combination of at least eight mixed characters.

Managing the USB Token

Go to the GeoTrust driver installation directory. Run `CIPUtils.exe` to manage the content of USB token.

6.6.2 Using the USB Token

The Sign MIDlet Suite dialog provides access to the USB token. With the USB token attached and the driver installed, select File > Load keystore > from USB Token (or, type Ctrl -T). If the USB token is password protected, you are prompted to enter a password. When the token is properly loaded all aliases and key details are listed. You can then select keys and use them for signing as described in Section 6.3, “Signing a MIDlet Suite” on page 6-4.

When you attempt to load a keystore from the USB token you might see an error message.

- If the USB token cannot be accessed, you might see the error “USB token or driver might be unplugged or invalid.“ Make sure the token is plugged in. You might need to try different USB ports on your machine.

- If the native library is not found, you are prompted to enter the path to the DLL library installed with the driver. It is usually in the directory in which you installed the driver.

CHAPTER 7

Using the Wireless Messaging API

The Sun Java™ Wireless Toolkit for CLDC supports the Wireless Messaging API (WMA) with a sophisticated simulation environment. WMA 1.1 (JSR 120) enables MIDlets to send and receive Short Message Service (SMS) or Cell Broadcast Service (CBS) messages. WMA 2.0 (JSR 205) includes support for MMS messages as well.

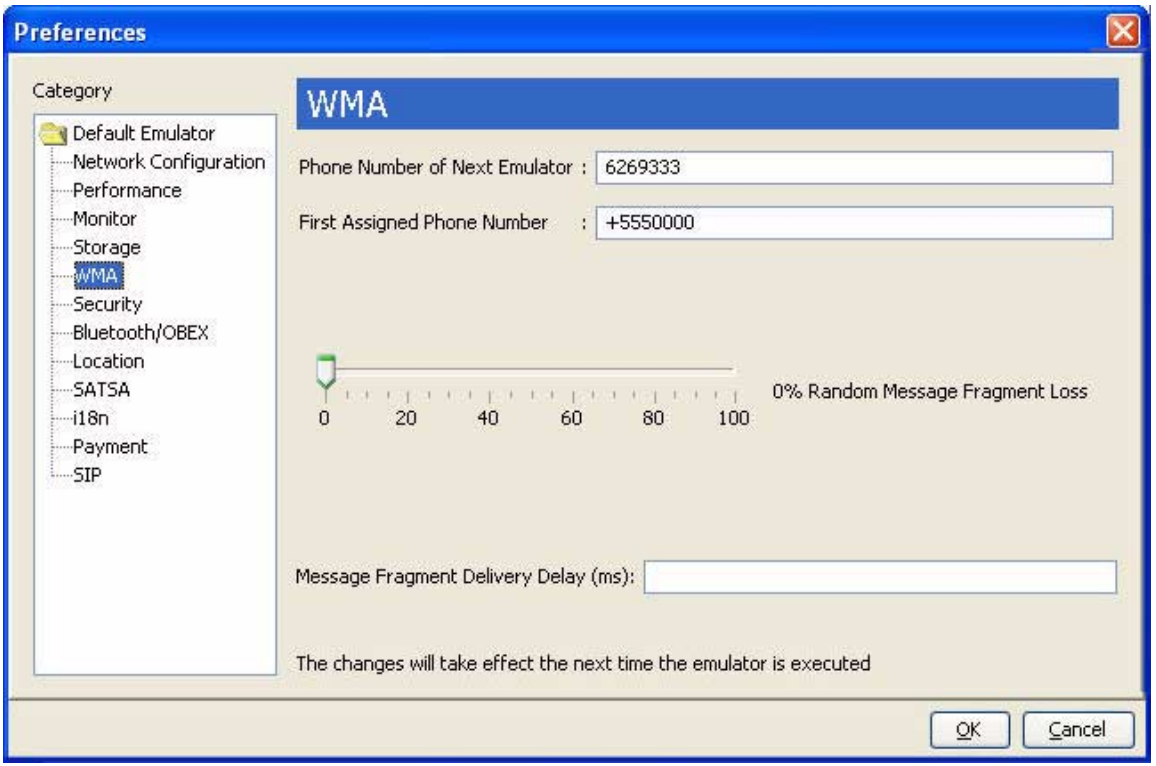
This chapter describes the tools you can use to develop WMA applications. It begins by showing how to configure the emulator's support of WMA. Next, it describes the WMA console, a handy utility for testing WMA applications. The chapter concludes with a brief description of the network monitor's WMA support.

7.1 Setting Emulator Phone Numbers

Each running instance of the emulator has a simulated phone number that is shown in the title bar of the emulator window. The phone numbers are important because they are used as addresses for WMA messages. By default, the first emulator instance has a phone number of +555000. Subsequent instances of the emulator have unique numbers in ascending order: +5550001, +5550002, +5550003, etc.

You can affect the assigned phone numbers by choosing Edit > Preferences and selecting WMA in the left pane.

FIGURE 7-1 Setting WMA Preferences



The Phone Number of Next Emulator field is just what it sounds like. If you fill in a number for this field, the next emulator instance will have that number.

If the Phone Number of Next Emulator is already in use, or if the field is blank, then the First Assigned Phone Number is used for the next emulator instance. Subsequent instances count up.

For example, for the Phone Number of Next Emulator, suppose you enter +6269333, and for the First Assigned Phone Number you enter +5550000. If you launch four emulator instances, their numbers are +6269333, +5550000, +5550001, and +5550002.

7.2 Simulating an Unreliable Network

Long messages are sent by splitting them, sending the fragments separately, and reassembling the fragments on the receiving end. You can simulate some of the hazards of the wireless network in the Sun Java™ Wireless Toolkit for CLDC. As before, choose Edit > Preferences and select WMA.

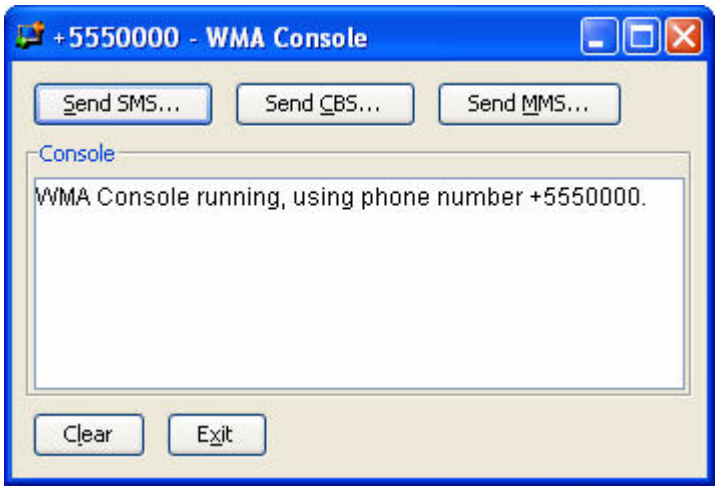
If you want the toolkit to lose some message fragments, adjust the Random Message Fragment Loss slider to the desired percentage. To simulate a delay between the time message fragments are sent and received, enter the delay in milliseconds in the Message Fragment Delivery Delay field.

7.3 Sending Messages With the WMA Console

The WMA console is a handy utility that enables you to send and receive messages. You can, for example, use the WMA console to send SMS messages to a MIDlet running on the emulator.

To launch the WMA console, choose File > Utilities. Click on WMA Console and press Launch.

FIGURE 7-2 The WMA Console



7.3.1 Sending a Text SMS Message

To send a text SMS message, click Send SMS The send window appears.

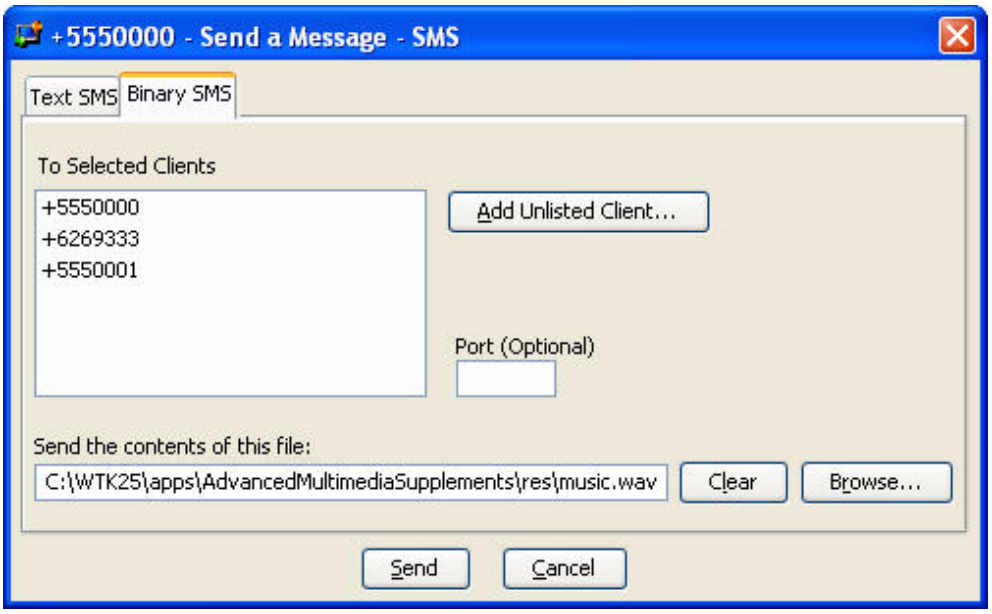
FIGURE 7-3 Sending a Text Message

The window automatically lists the phone numbers of all running emulator instances. Select a destination (Control-click to select multiple destinations) and enter a port number if you wish. Type your message and click Send.

7.3.2 Sending a Binary SMS Message

You can use the WMA console to send the contents of a file as a binary message. Click Send SMS to bring up the send window. Click the Binary SMS tab.

FIGURE 7-4 Sending a Binary Message



Selecting recipients is the same as for sending text SMS messages. You can type in the path of a file directly, or click Browse to open a file chooser.

7.3.3 Sending Text or Binary CBS Messages

Sending CBS messages is similar to sending SMS messages except that you don't need to choose recipients. To send a text or binary CBS message, click Send CBS in the WMA console. The Send window appears.

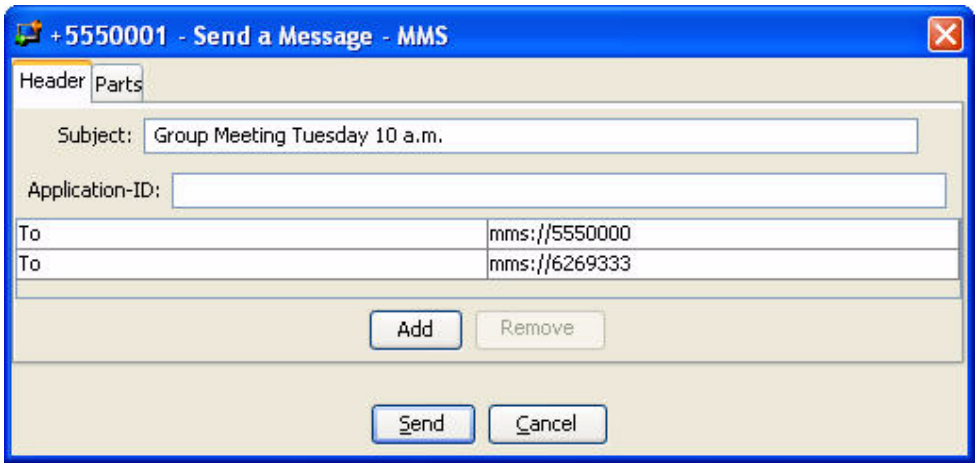
FIGURE 7-5 Sending CBS Messages

7.3.4 Sending MMS Messages

MMS messages consist of one or more files, usually images or sounds. MMS message can be sent to multiple recipients. To send an MMS message from the WMA console, click the Send MMS button.

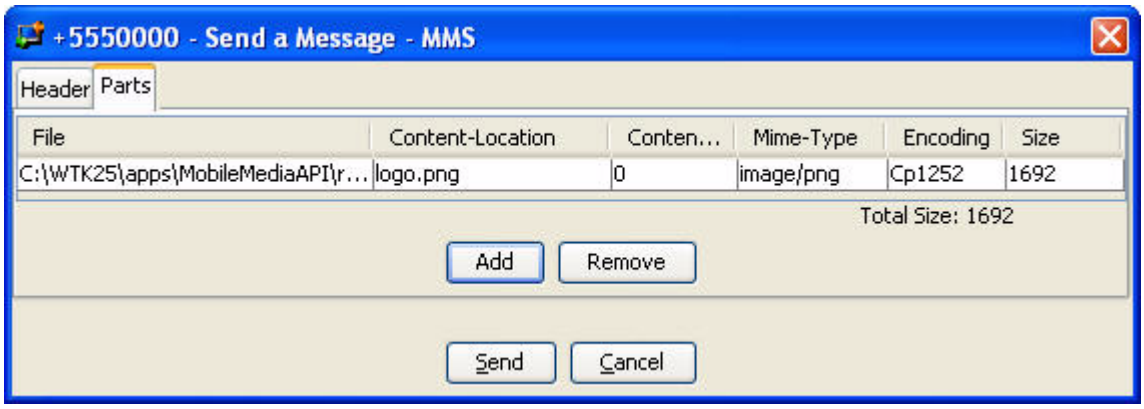
The window for composing MMS messages has two tabs, one for recipients and one for content. Begin by filling in a subject and recipient. If you wish to add more recipients, click the Add button. For example, to send a message to a running emulator whose number is +5550001, fill in the To line as `mms://+5550001`. To remove a recipient, first select its line, then click Remove.

FIGURE 7-6 Adding Recipients for an MMS Message



To add media files to the message, click the Parts tab. Click Add to add a part to the message. To remove a part, select it and press Remove.

FIGURE 7-7 Adding Parts to an MMS Message



7.4 Receiving Messages in the WMA Console

The WMA console can also receive messages. The WMA console window has its own phone number in the title bar. You can send messages to the WMA console from your applications running on the emulator.

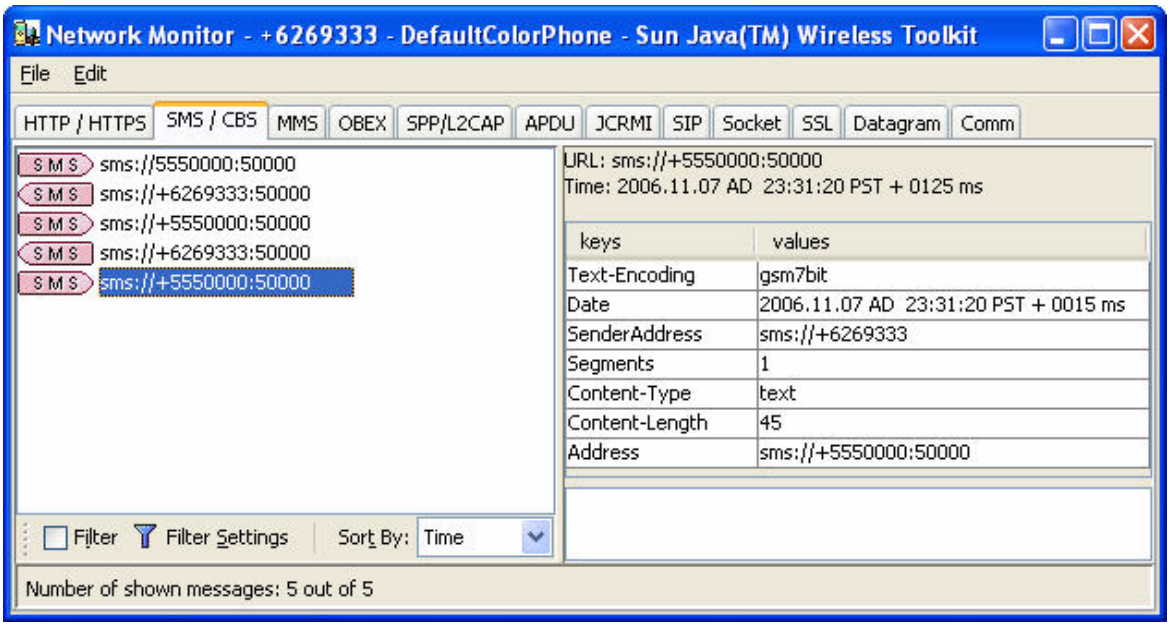
Received messages are shown in the WMA console's text area.

7.5 Using the Network Monitor with WMA

The network monitor is fully described in Chapter 5. You can use the network monitor to track WMA messages that are sent to or from the emulator.

Click the SMS/CBS or MMS tabs to see WMA messages. Information about the messages and their fragments is shown in the left pane of the network monitor. Click a message or message fragment to see its details in the right pane.

FIGURE 7-8 Using the Network Monitor to View a WMA Message



CHAPTER 8

Using the Mobile Media API

JSR 135, the Mobile Media API (MMAPI), provides a standard API for rendering and capturing time-based media, like audio or video. The API is designed to be flexible with respect to the media formats, protocols, and features supported by various devices. For information on programming with MMAPI, see the following articles:

- *Mobile Media API Overview*

http://developers.sun.com/techtopics/mobility/apis/articles/mmapi_overview/

- *The J2ME Mobile Media API*

http://developers.sun.com/techtopics/mobility/midp/articles/mmapioverview/

8.1 Supported Formats and Protocols

The emulator's MMAPI implementation supports the following media types.

TABLE 8-1 Supported MMAPI Media Formats

MIME Type	Description
audio/amr	Adaptive Multi-Rate
audio/midi	MIDI files
audio/sp-midi	Scalable Polyphony MIDI
audio/x-tone-seq	MIDP 2.0 tone sequence
audio/x-wav	WAV PCM sampled audio
image/gif	GIF 89a (animated GIF)
video/mpeg	MPEG video
video/vnd.sun.rgb565	Video capture

8.2 Adaptive Multi-Rate (AMR) Content

The Sun Java™ Wireless Toolkit for CLDC simulates support for Adaptive Multi-Rate (AMR) content (http://www.ietf.org/rfc/rfc3267.txt). Although the toolkit cannot decode AMR content, the implementation returns a player for AMR content when requested.

8.2.1 Windows

On Windows, AMR files are converted to regular WAVE files and passed to Qsound. Because the Windows version interfaces with the 3GPP implementation, you do not have to do anything to ge t AMR files to play.

8.2.2 Linux

The AMR support for Linux is based on the 3GPP AMR Narrow Band (AMR-NB) Reference Implementation decoder and the SOX audio processor (AMR Wide Band is not supported).

8.2.2.1 Enabling AMR Support

Follow these steps to enable AMR support.

1. **Get the AMR-NB RI provided by 3GPP.**

   There are several versions available. One can be found here:

   http://www.3gpp.org/ftp/Specs/archive/26_series/26.073/26073-530.zip

2. **Open the makefile contained in the package.**

   Find the line starting with 'CFLAGS = '.

   a. **Add the** `DMMS_IO` **option.**

   b. **Remove the** `pedantic-errors` **option.**

   Save and compile. If you have problems, try removing the the O3 (or O2) optimization flags as well (the binaries will be nearly twice the size).

3. **To build the RI , enter:** `make VAD=VAD1`

   After the compilation you should have a binary file named `decoder`. That's the AMR-NB decoder itself.

4. **Set the environment variable `AMR_DECODER` to point to the path to the decoder.**

   For example, if `~/amr` is the path to the decoder, specify:

   ```
   export AMR_DECODER=~/amr/decoder
   ```

5. **Set execution access rights for the `decoder` file.**

   In the same directory as the decoder, type:

   ```
   chmod 555 ./decoder
   ```

6. **Test the decoder as follows:**

   Run the MobileMediaAPI demo Simple Player as described in Section A.13, "MobileMediaAPI" on page A-31. From the main menu select Simple Player and then 'AMR Narrow Band [jar]'. Don't forget to turn your speakers on.

8.2.2.2 AMR Format Support

You may also encounter problems with some "clones" of the AMR-NB. There appear to be at least two file formats for AMR-NB codec data:

- `.AMR` files stored in "AMR File Storage Format"

  This is specified in `draft-ietf-avt-rtp-amr-10.txt`, Sec. 6.2, which is included with the Ericsson AMR converter tool. (The draft is an early form of RFC 3267). These files are handled by the Ericsson AMR tool and Nokia Series 60 phones. They have a header of `#!AMR\n` and they're encoded in big-endian.

- `.COD` files stored in "AMR Interface Format 2"

  This format is specified in 3GPP TS 26.101, Appendix A. These are coded and decoded by the 3GPP TS 26.104 floating point reference codec source package. These files have no header and are encoded in little-endian.

To convert from between these formats try this Python script:

http://www.connactivity.com/~eaw/amrwork/amrconv.py

8.3 Using `MediaControlSkin`

The Sun Java™ Wireless Toolkit for CLDC comes with an emulator skin, `MediaControlSkin`, that is focused on multimedia playback and control. The skin includes buttons with symbols representing play, stop, volume up and volume down, and other commands. To see the usefulness of `MediaControlSkin`, try it out with the `MobileMediaAPI` demonstration application.

8.4 Media Capture

The Sun Java™ Wireless Toolkit for CLDC emulator supports audio and video capture. Audio capture is supported by using the capture capabilities of the system upon which the emulator runs.

Video capture is supported by simulating a camera input.

Consult the `MobileMediaAPI` example application for details and source code that demonstrates how to capture audio and video.

8.5 Well-Behaved MIDlets

MIDlets have a life cycle that is defined in the MIDP specification. MIDlets can be paused by events such as incoming phone calls. A well-behaved MIDlet releases important device resources when it is paused and reallocates or restarts those resources when the MIDlet is resumed. In the MMAPI arena, stop any `Players` that are rendering content when a MIDlet is paused.

The Sun Java™ Wireless Toolkit for CLDC prints a message to the console if you pause a MIDlet and it does not stop its running `Players`. You can test this feature using the Pausing Audio Test MIDlet in the `MobileMediaAPI` demonstration application. See Appendix A for details.

The warning message is printed once only for each running emulator.

8.6 Ring Tones

MMAPI can be used to play ring tones, as demonstrated in Section A.13.1, "Simple Tones" on page A-31 and Section A.13.2, "Simple Player" on page A-32. Several ring tone formats are in common use. You can download ring tones or create your own.

8.6.1 Download Ring Tones

Ring tone files can be downloaded from many internet sites, including the following:

- http://www.surgeryofsound.co.uk/
- http://www.convertyourtone.com/
- http://www.filmfind.tv/ringtones/

8.6.2 Ring Tone Formats

This section provides samples of several formats

- RTTTL, the Ringing Tones text transfer language format, is explained at

  http://www.convertyourtone.com/rtttl.html

- Nokia Composer

  This is a rendition of Beethoven's 9th symphony in Nokia Composer format:

  ```
  16g1,16g1,16g1,4#d1,16f1,16f1,16f1,4d1,16g1,16g1,16g1,16#d1,
  16#g1,16#g1,16#g1,16g1,16#d2,16#d2,16#d2,4c2,16g1,16g1,16g1,
  16d1,16#g1,16#g1,16#g1, 16g1,16f2,16f2,16f2,4d2
  ```

- Ericsson Composer

  Beethoven's Menuett in G:

  ```
  a b + c b + c b + c b + C p + d a B p + c g A
  p f g a g a g a g A p b f G p a e F
  ```

  Beethoven's 9th symphony theme:

  ```
  f f f # C # d # d # d C p f f f # c # f #f # f f +# c + #
  c + # c # A ff f c # f # f # f f + # d + # d + # d
  ```

- Siemens Composer Format

  Inspector Gadget theme:

```
C2(1/8) D2(1/16) Dis2(1/8) F2(1/16) G2(1/8)
P(1/16) Dis2(1/8) P(1/16) Fis2(1/8) P(1/16)
D2(1/8) P(1/16) F2(1/8) P(1/16) Dis2(1/8)
P(1/16) C2(1/8) D2(1/16) Dis2(1/8) F2(1/16)
G2(1/8) P(1/16) C3(1/8) P(1/16) B2(1/2) P(1/4)
C2(1/8) D2(1/16) Dis2(1/8) F2(1/16) G2(1/8) P(1/16)
Dis2(1/8) P(1/16) Fis2(1/8) P(1/16) D2(1/8) P(1/16)
F2(1/8) P(1/16) Dis2(1/8) P(1/16) C3(1/8) B2(1/16)
Ais2(1/8) A2(1/16) Gis2(1/2) G2(1/8) P(1/16) C3(1/2)
```

- Motorola Composer

  Beethovens 9th symphony:

```
4 F2 F2 F2 C#4 D#2 D#2 D#2 C4 R2 F2 F2 F2 C#2 F#2 F#2
F#2 F2 C#+2 C#+2 C#+2 A#4 F2 F2 F2 C2 F#2 F#2 F#2 F2
D#+2 D#+2 D#+2
```

- Panasonic Composer

  Beethovens 9thsymphony:

```
444** 444** 444** 1111* 4444** 4444** 4444** 111*
0** 444** 444** 444** 1111** 4444** 4444** 4444**
444** 11** 11** 11** 6666* 444** 444** 444** 111**
4444** 4444** 4444** 444** 22** 22** 22**
```

- Sony Composer

  Beethovens 9th sympony:

```
444****444****444****111#*****444#****444#****444#****
111*****(JD)0000444****444****444****111#****444#****
444#****444#****444****11#****11#****11#****666#*****
444****444****444****111****444#****444#****
444#****444****22#****22#****22#****
```

CHAPTER 9

Working With Mobile Graphics

This chapter provides a brief overview of working with graphics content. The Sun Java™ Wireless Toolkit for CLDC offers three APIs that provide comprehensive capabilities for interactive 2D and 3D graphics:

- The Mobile 3D Graphics API for J2ME, JSR 184, provides 3D graphics capabilities with a low-level API and a high-level scene graph API. This chapter provides a brief overview and general guidelines for working with JSR 184.
- The Scalable 2D Vector Graphics API for J2ME, JSR 226, supports rendering sophisticated and interactive 2D content.
- Java Bindings for OpenGL® ES, JSR 239, provides a Java language interface to the open standard OpenGL® ES graphics API.

9.1 Using the Mobile 3D Graphics API

JSR 184 is a specification that defines the Mobile 3D Graphics (M3G) API for the J2ME. This API provides 3D functionality in a compact package that's appropriate for CLDC/MIDP devices. The API provides two methods for displaying 3D graphics content. The *immediate mode* API makes it possible for applications to directly create and manipulate 3D elements. Layered on top of this is a *scene graph* API, also called *retained mode*, that makes it possible to load and display entire 3D scenes that are designed ahead of time. Applications are free to use whichever approach is most appropriate or to use a combination of the retained mode and immediate mode APIs. The JSR 184 specification also defines a file format (`.m3g`) for scene graphs.

For more information, consult the JSR 184 specification at `http://jcp.org/en/jsr/detail?id=184`.

JSR 184 provides a standard API for CLDC/MIDP devices, enabling a new generation of 3D applications. The immediate mode API, in turn, is compatible with OpenGL ES, a standard lightweight API for 3D graphics. See `http://khronos.org/` for more information on OpenGL ES.

9.1.1 Immediate Mode

Immediate mode is appropriate for applications that generate 3D graphics content algorithmically, like scientific visualizations or statistical graphs. The application creates 3D objects and manipulates them directly.

For an example of immediate mode, see the `Life3D` MIDlet in the `Demo3D` example application.

9.1.2 Retained Mode

Most applications, particularly games, use the retained mode or scene graph API. In this approach, a graphic designer or artist uses 3D modeling software to create a scene graph. The scene graph is saved in the JSR 184 file format. The scene graph file is bundled with the application. At runtime, the application uses the scene graph API to load and display the file.

Applications can manipulate parts of a loaded scene graph to animate characters or create other effects. The basic strategy is to do as much work as possible in the modeling software. At runtime, the application can grab and manipulate parts of the scene graph, which can also include paths for animation or other effects.

For an example of retained mode, see the `retainedmode` MIDlet in the `Demo3D` example application.

9.1.3 Trading Quality for Speed

One of the challenges of MIDP development is the constrained environment of typical devices. Compared to desktop computers, MIDP devices have slow processors and little memory. These challenges extend into the arena of 3D graphics. To accommodate a wide variety of implementations, the JSR 184 specification provides various mechanisms to make the display of a 3D scene as efficient as possible.

One approach is *scoping*, a technique where you tell the 3D graphics implementation when objects are not going to interact with each other. For example, if you defined a scene graph for a house, you could use scoping to specify that the light in the

basement doesn't affect the appearance of the bedroom on the second floor. Scoping makes the implementation's job easier by reducing the number of calculations required to show a scene.

In general, however, the best way to improve the rendering speed of 3D scenes is to make some compromises in quality. The Mobile 3D Graphics API includes *rendering hints* so that applications can suggest how the implementation can compromise quality to improve rendering speed.

9.1.4 Creating Mobile 3D Graphics Content

Most mobile 3D applications use scene graphs in resource files to describe objects, scenes, and characters. Usually it is not programmers but graphic designers or artists who create the scene graphs, using standard 3D modeling tools.

Several vendors offer tools for authoring content and converting files to the JSR 184 format. Superscape (`http://superscape.com/`) is one such vendor.

Because it is relatively difficult to create and manipulate 3D graphics content in an application using the immediate mode API, most applications rely as much as possible on a scene graph file. By putting as much as possible into the scene graph file at design time, the application's job at runtime is considerably simplified.

9.2 Rendering Scalable Vector Graphics Content

Scalable Vector Graphics (SVG) is a standard defined by the World Wide Web Consortium. It is an XML grammar for describing rich, interactive 2D graphics.

The Sun Java™ Wireless Toolkit for CLDC emulator support JSR 226, the Scalable 2D Vector Graphics API for J2ME. JSR 226 is a Java ME API to load, manipulate, render and play SVG content. SVG Tiny is an compact yet powerful XML format for describing rich, interactive, animated 2D content.

While it is possible to produce SVG content with a text editor, most people prefer to use an authoring tool. Here are three possibilities:

- **BeatWare Mobile Designer -** `http://www.beatware.com/products/md_golive.html`
- **Ikivo Animator -** `http://www.ikivo.com/animator/`
- **Adobe Illustrator CS2 -** `http://www.adobe.com/products/illustrator/main.html`

Java ME applications using SVG content can create graphical effects that adapt to the display resolution and form factor of the user's display.

SVG images can be animated in two ways. One is to use declarative animations. The other is to repeatedly modify the SVG image parameters (such as color or position), through API calls. Section A.21.3, “Play SVG Animation” on page A-52 illustrates declarative animation.

9.3 OpenGL® ES Overview

JSR 239 defines the Java programming language bindings for two APIs, OpenGL® for Embedded Systems (OpenGL® ES) and EGL. OpenGL® ES is a standard API for 3D graphics, a subset of OpenGL®, which is pervasive on desktop computers. EGL is a standard platform interface layer. Both OpenGL® ES and EGL are developed by the Khronos Group (`http://khronos.org/opengles/`).

While JSR 184 (which is object oriented) requires high level functionality, OpenGL® is a low level graphics library that is suited for accessing hardware accelerated 3D graphics. Explore the OpenGLESDemo sample project code.

CHAPTER 10

Using the PIM and FileConnection APIs

The Sun Java™ Wireless Toolkit for CLDC supports JSR 75, the PDA Optional Packages (PDAP) for the J2ME Platform. JSR 75 includes two independent APIs:

- The FileConnection optional package allows MIDlets access to a local device file system.
- The Personal Information Management (PIM) optional package includes APIs for manipulating contact lists (address book), calendars, and to-do lists.

This chapter describes how the Sun Java™ Wireless Toolkit for CLDC implements the FileConnection and PIM APIs.

10.1 FileConnection API

On a real device, the FileConnection API typically provides access to files stored in the device's memory or on a memory card.

In the Sun Java™ Wireless Toolkit for CLDC emulator, the FileConnection API enables MIDlets to access files stored on your desktop computer's hard disk.

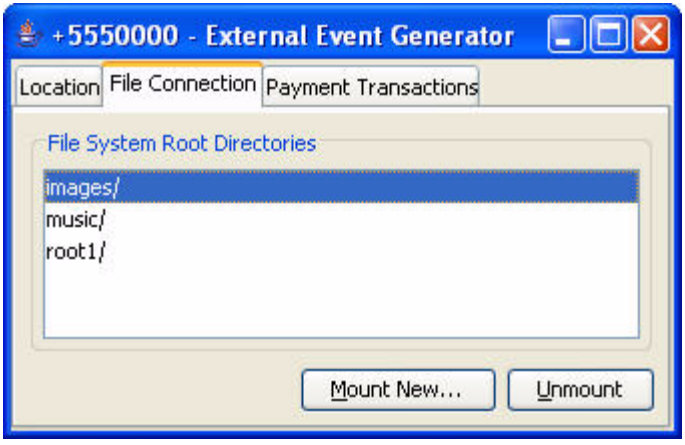
The files that can be accessed using FileConnection are stored in subdirectories of *workdir*`\appdb\`*skin*`\filesystem`. For example, the `DefaultColorPhone` emulator skin comes with a root directory installed called `root1`, which contains a file called `Readme` and an empty directory named `photos`. The full path of the file is:

Windows:	*workdir*`\appdb\`*skin*`\filesystem\root1\photos`
Linux:	*workdir*`/appdb/`*skin*`/filesystem/root1/photos`

Note – If multiple instances of the same emulator skin run simultaneously, the Sun Java™ Wireless Toolkit for CLDC generates unique file paths for each one. For instance, the first directory is named `DefaultColorPhone` and the second instance is named `DefaultColorPhone1`.

Each subdirectory of `filesystem` is called a *root*. The Sun Java™ Wireless Toolkit for CLDC provides a mechanism for managing roots. While the emulator is running, choose MIDlet > External events from the emulator window's menu. A utility window for adding and removing roots appears.

FIGURE 10-1 Managing File System Roots



The mounted roots and their contents are available to applications using the FileConnection API.

To add a new root directory, click Mount New and fill in a name for the directory. To make a directory inaccessible to the FileConnection API, select it in the list and click Unmount.

10.2 The PIM API

The Sun Java™ Wireless Toolkit for CLDC emulator stores contact, calendar, and to-do information in standard files on your desktop computer's hard disk. All information is stored in *workdir*\appdb*skin*\pim. This directory is shared by all running emulators. Lists are stored in subdirectories of the contacts, events, and todo directories. For example, a contact list called Contacts is contained in:

Windows:	*workdir*\appdb*skin*\pim\contacts\Contacts
Linux:	*workdir*/appdb/*skin*/pim/contacts/Contacts

Inside the list directory, items are stored in vCard (.vcs) or vCalendar (.vcf) format (see http://www.imc.org/pdi/). Contacts are stored in vCard format, while calendar and to-do items are both stored in vCalendar format.

CHAPTER 11

Using the Bluetooth and OBEX APIs

The Sun Java™ Wireless Toolkit for CLDC emulator supports JSR 82, the Java APIs for Bluetooth. The emulator is fully compliant with version 1.1 of the specification, which describes integration with the push registry. JSR 82 includes two independent APIs:

- The Bluetooth API provides an interface to Bluetooth wireless networking, including device discovery and data exchange.
- The OBEX API allows applications to use the Object Exchange (OBEX) protocol over Bluetooth or other communication channels.

This chapter describes how the Sun Java™ Wireless Toolkit for CLDC implements the Bluetooth and OBEX APIs.

11.1 Bluetooth Simulation Environment

The Sun Java™ Wireless Toolkit for CLDC emulator enables you to develop and test application that use Bluetooth without having actual Bluetooth hardware. The toolkit simulates a Bluetooth environment for running emulators. Multiple emulator instances can discover each other and exchange data using the Bluetooth API.

For an example, see the documentation of `BluetoothDemo` in Appendix A.

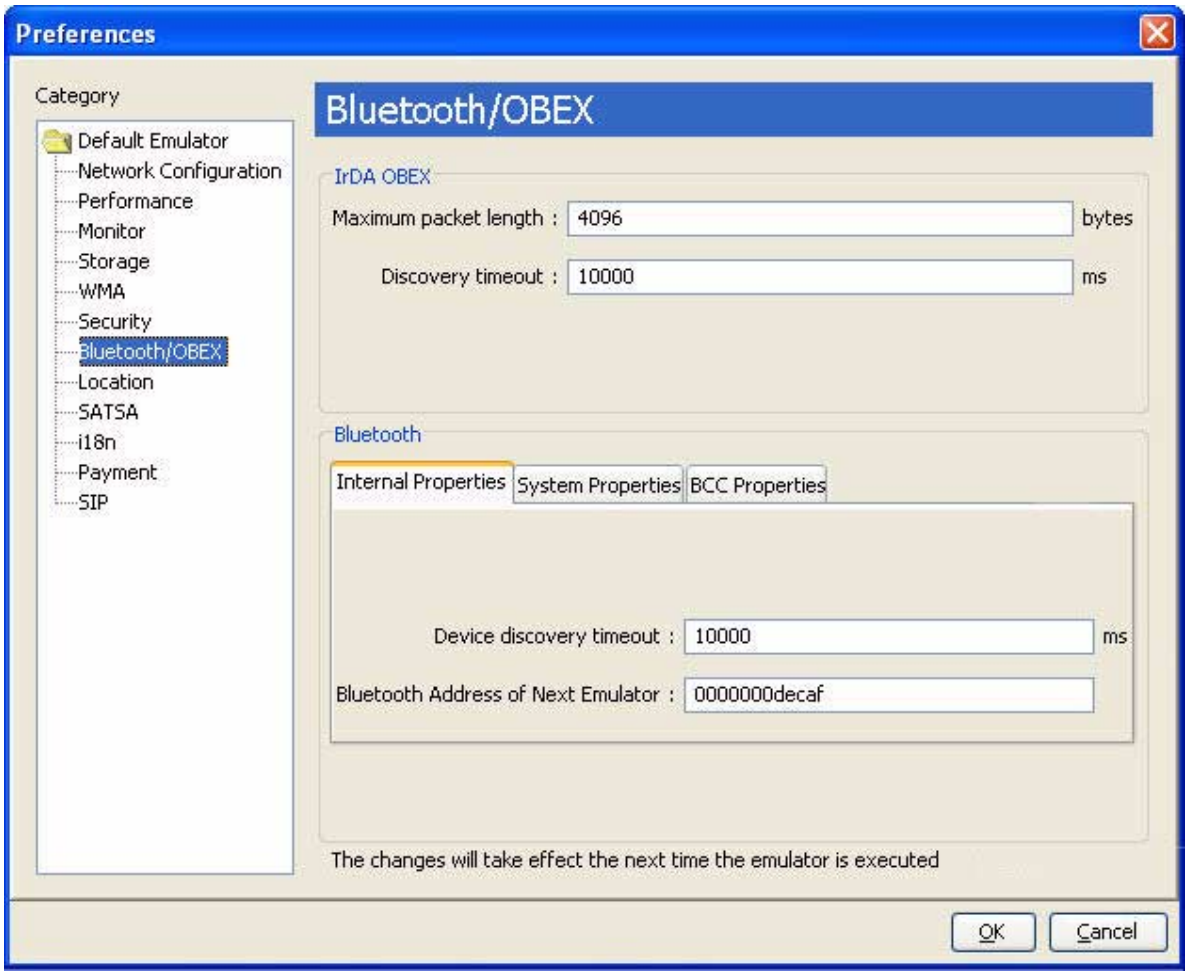
11.2 OBEX Over Infrared

The Sun Java™ Wireless Toolkit for CLDC implements OBEX transfer over simulated Bluetooth and infrared connections. The simulated infrared connection follows the IrDA standard (see http://www.irda.org/). Simulated infrared transfers can take place between multiple running emulators.

11.3 Setting OBEX and Bluetooth Preferences

The Sun Java™ Wireless Toolkit for CLDC enables you to configure the Bluetooth and OBEX simulation environment. Choose Edit > Preferences and select Bluetooth/OBEX to display the following window.

FIGURE 11-1 Bluetooth and OBEX Preferences



11.3.1 OBEX Preferences

Devices using IrDA in the real world discover other devices by listening. You can configure how long the Sun Java™ Wireless Toolkit for CLDC emulator waits to discover another device using the Discovery timeout field in the IrDA OBEX section of the preferences window. Enter a value in milliseconds.

At the API level, the discovery timeout value determines how long a call to `Connector.open("irdaobex://discover...")` blocks before it returns or throws an exception.

The maximum packet length affects how much data is sent in each packet between emulators. Shorter packet values result in more packets and more packet overhead.

11.3.2 Bluetooth Internal Properties

In the Bluetooth section of the preferences window, the Device discovery timeout is the amount of time, in milliseconds, the emulator waits while attempting to locate other devices in the simulated Bluetooth environment.

Bluetooth Address of Next Emulator is the Bluetooth address to be assigned to the first emulator instance. Subsequent instances of the emulator receive an automatically generated address.

11.3.3 Bluetooth System Properties

The System Properties tab in the Bluetooth section of the preferences contains properties that can be retrieved in an application using the `getProperty()` method in `javax.bluetooth.LocalDevice`.

The Bluetooth properties are fully described in the JSR 82 specification.

11.3.4 Bluetooth BCC Properties

The Bluetooth Control Center (BCC) controls Bluetooth settings. Some devices might provide a GUI to customize Bluetooth settings. In the Sun Java™ Wireless Toolkit for CLDC, the BCC is configured using the BCC Properties tab of the Bluetooth preferences. The properties are as follows.

TABLE 11-1 BCC Properties

Property	Description
Enable Bluetooth support	If this property is disabled, `LocalDevice.getLocalDevice()` throws a `BluetoothStateException` and no connections can be created. This is useful to test the behavior of your application on devices that support JSR 82 but might have the Bluetooth feature turned off.
Device is discoverable	Indicates whether or not this emulator can be discovered by other emulators.
Friendly name	A human-readable name for the emulator in the simulated Bluetooth environment. If the name is left blank, the emulator does not support the friendly name feature.

TABLE 11-1 BCC Properties (Continued)

Property	Description
Encryption	Determines whether connection encryption is supported (`on`) or not (`off`). In addition, the `force` settings means all connections must be encrypted. See the documentation for `RemoteDevice`'s `encrypt()` method for details.
Authorization	Similar to the Encryption property. See `RemoteDevice`'s `authorize()` method.
Authentication	Similar to Encryption and Authorization. See `RemoteDevice`'s `authenticate()` method.

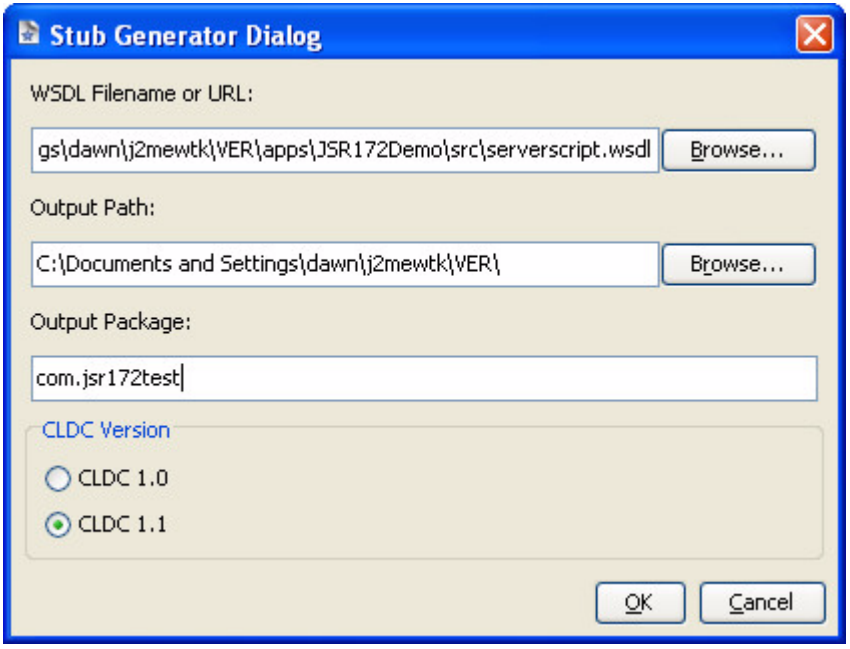
CHAPTER 12

Using Web Services

The Sun Java™ Wireless Toolkit for CLDC emulator supports JSR 172, the J2ME Web Services Specification. JSR 172 provides APIs for accessing web services from mobile applications. It also includes an API for parsing XML documents.

The Sun Java™ Wireless Toolkit for CLDC provides a stub generator that automates creating source code for accessing web services. To get to the stub generator, choose File > Utilities. Click Stub Generator and press Launch.

FIGURE 12-1 Web Services Stub Generator



In the field WSDL Filename or URL supply the path to the Webservices Description Language (WSDL) file for the web service you want to access. The Output Path indicates the location where you want the stub files to be placed. Output Package indicates the Java programming language package name for the stub files. Finally, choose whether you want to generate CLDC 1.0 or CLDC 1.1 stubs.

Press OK to generate the stub files.

CHAPTER 13

Using the Location API

The JSR 179 Location API gives applications the opportunity to use a device's location capabilities. For example, some devices include Global Positioning System (GPS) hardware. Other devices might be able to receive location information from the wireless network. The Location API provides a standard interface to location information, regardless of the underlying technique.

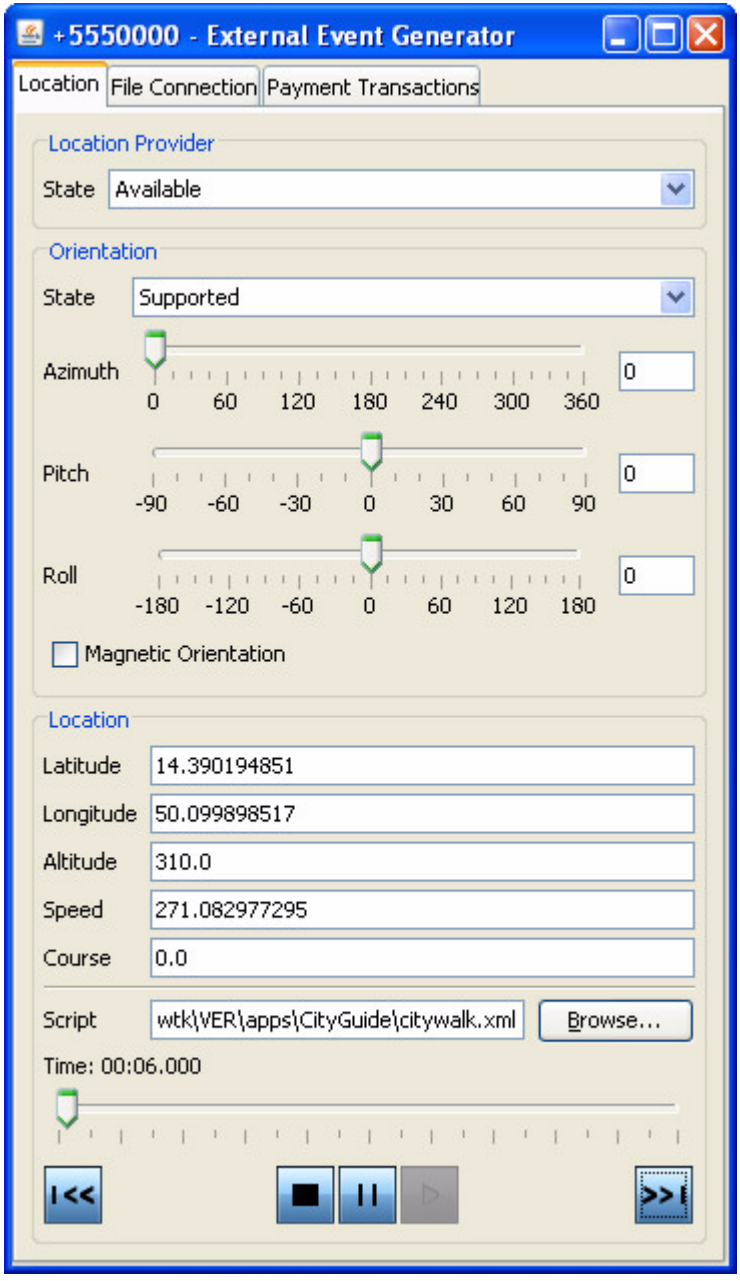
In the Location API, a *location provider* encapsulates a positioning method and supplies information about the device's location. The application requests a provider by specifying required criteria, such as the desired accuracy and response time. If an appropriate implementation is available, the application can use it to obtain information about the device's physical location.

The Sun Java™ Wireless Toolkit for CLDC includes a simulated location provider. You can use the emulator's External Events window to specify where the emulator should think it is located. In addition, you can configure the properties of the provider itself, and you can manage a database of landmarks.

13.1 Setting the Emulator's Location at Runtime

You can specify the simulated location of the emulator while it is running. To do this, choose MIDlet > External Events from the emulator window's menu. Click the Location tab. See FIGURE 13-1.

FIGURE 13-1 Controlling Location in the Emulator



In the Location area of the tab, you can fill in values for the latitude, longitude, altitude, speed, and course. Applications that use the Location API can retrieve these

values as the location of the emulator.

For more elaborate testing, you can set up a location script that describes motion over time. Location scripts are XML files consisting of a list of locations, called *waypoints*, and associated times. The Sun Java™ Wireless Toolkit for CLDC determines the current location of the emulator by interpolating between the points in the location script. Here, for example, is a simple location script that specifies a starting point (`time="0"`) and moves to a new point in ten seconds:

```
<waypoints>
  <waypoint time="0"
            latitude="14" longitude="50" altitude="310" />
  <waypoint time="10000"
            latitude="14.5" longitude="50.1" altitude="215" />
</waypoints>
```

The altitude measurement is in meters, and the time values are in milliseconds.

Use a text editor to create your location script. You can load it into the external event window by pressing the Browse button next to the Script field. Immediately below are controls for playing, pausing, stopping, and moving to the beginning and end of the location script. You can also drag the time slider to a particular point.

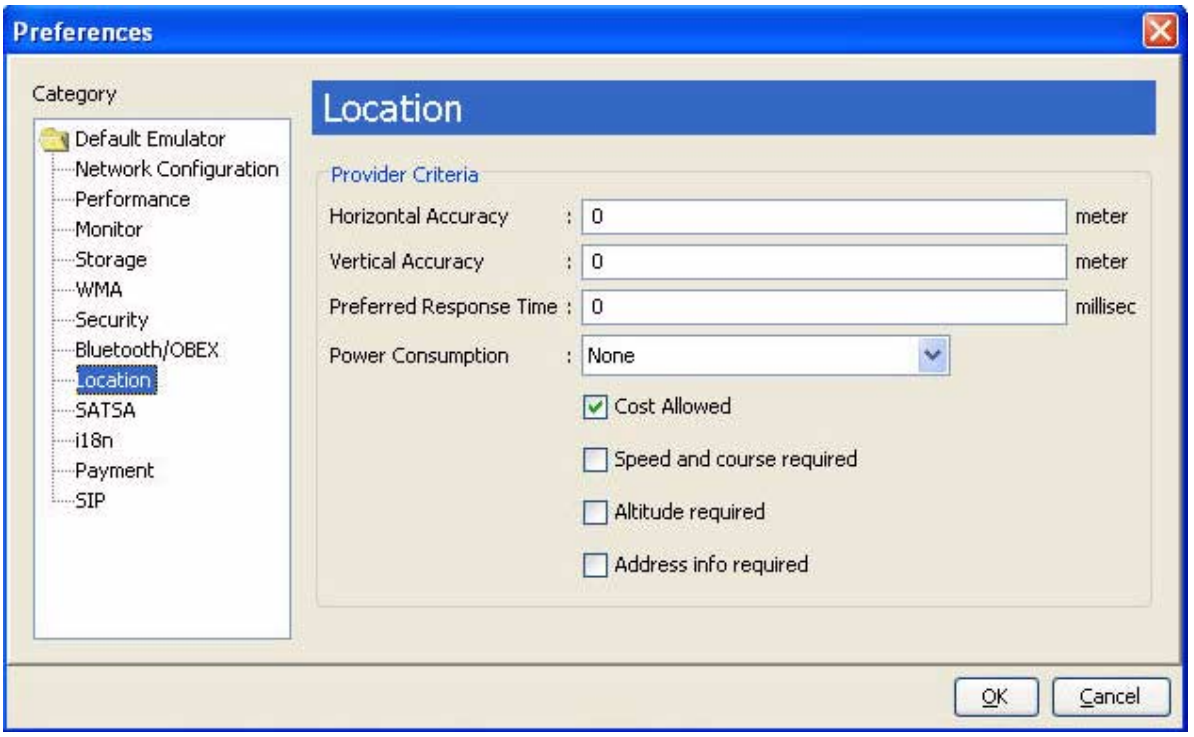
Some devices are also capable of measuring their orientation. To make this kind of information available to your application, change the State field in the Orientation box to Supported and fill in values for azimuth, pitch, and roll. The Magnetic Orientation checkbox indicates whether the azimuth and pitch measurements are relative to the Earth's magnetic field or relative to true north and gravity.

To test how your application handles unexpected conditions, try changing the State field in the Location Provider box to Temporarily Unavailable or Out of Service. When your application attempts to retrieve the emulator's location, an exception is thrown and you can see how your application responds.

13.2 Configuring the Location Provider

You can configure the properties of the Sun Java™ Wireless Toolkit for CLDC's location provider using the preferences. In the user interface, choose Edit > Preferences and click Location.

FIGURE 13-2 Configuring the Location Provider

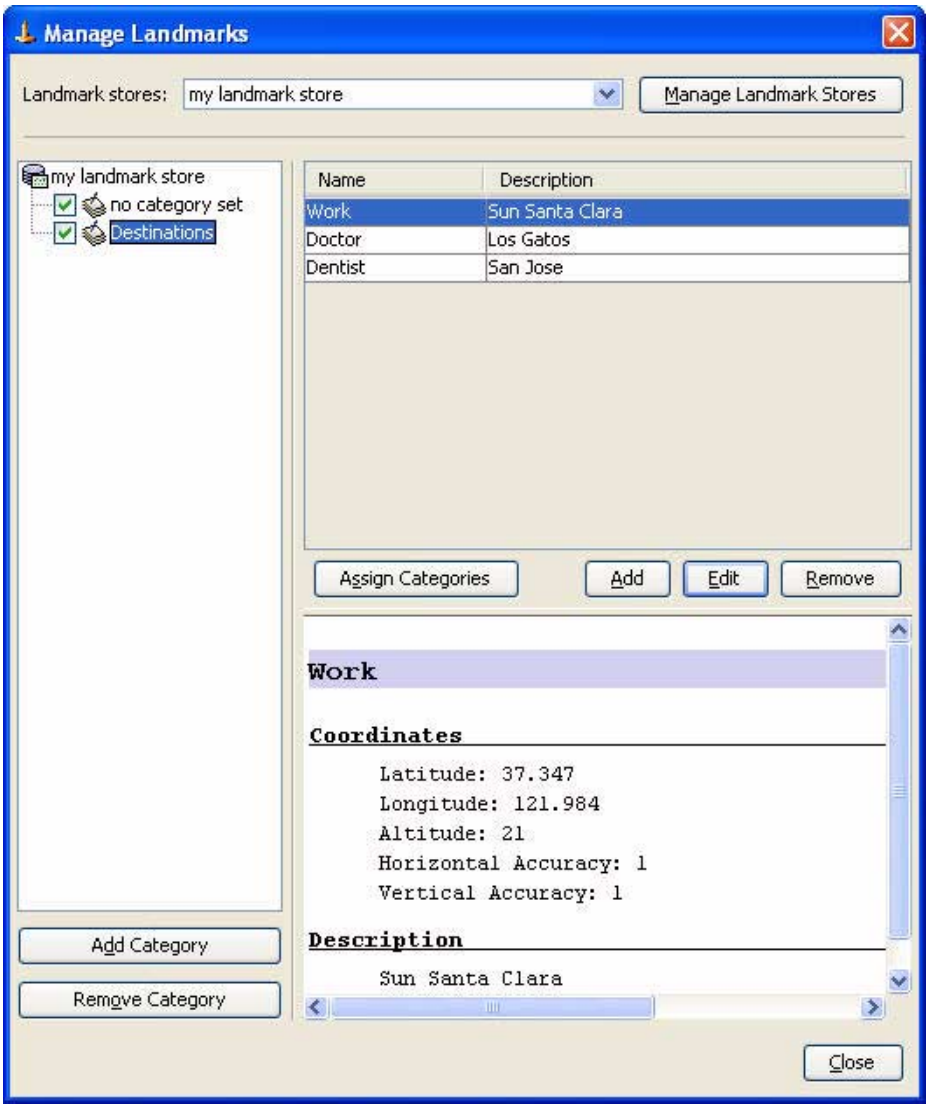


The fields in the Location tab enable you to specify the properties of the toolkit's built-in location provider. The properties you specify in the preferences correspond to the `Criteria` class applications use to request a location provider.

13.3 Setting Up Landmarks

The Sun Java™ Wireless Toolkit for CLDC emulator includes a *landmark store* system, just like many real devices. A landmark store is a collection of places with associated names and other information. To manage landmark stores, choose File > Utilities from the menu, select Manage Landmarks, and press Launch.

FIGURE 13-3 Landmark Manager



The landmark manager shows the content of a single landmark store. JSR 179 requires a minimum of one landmark store, and it is referred to as the default store. To select a different landmark store or create a new one, make a selection from the *Landmark stores* combo box at the top of the window.

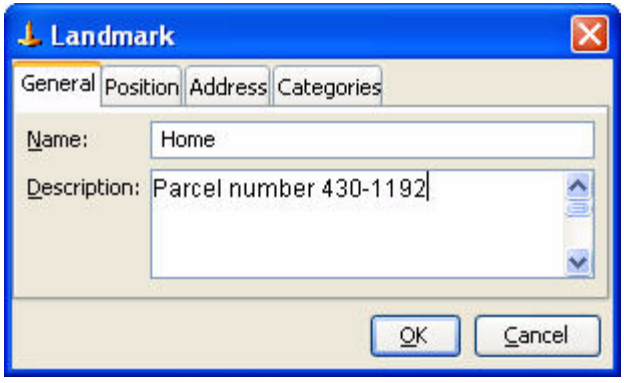
You can add or remove landmark stores by clicking the Manage Landmark Stores button. Landmark stores cannot be renamed.

Landmarks can be associated with *categories*, which are specific to a landmark store. The categories for the current landmark store are shown in the left pane of the window. You can add or remove categories using the buttons at the bottom of the list. Check off one or more of the categories if you would like to see only the matching landmarks. You can also check *no category set* to see landmarks with no associated categories.

The right pane of the landmark manager lists the landmarks in the current landmark store. Click a landmark to see its details listed in the bottom part of the right pane.

To add a new landmark, click Add and fill in the fields as appropriate. Click Edit to change the currently selected landmark. Finally, press Remove to remove the currently selected landmark.

FIGURE 13-4 Adding or Editing a Landmark



You can also use the Assign Categories in the main window to specify the categories for a landmark.

CHAPTER 14

Using SATSA

The Security and Trust Services APIs (SATSA) provide smart card access and cryptographic capabilities to applications running on small devices. JSR 177 (the SATSA specification) defines four distinct APIs as optional packages:

- **SATSA-APDU -** Enables applications to communicate with smart card applications using a low-level protocol.
- **SATSA-JCRMI -** Provides an alternate method for communicating with smart card applications using a remote object protocol.
- **SATSA-PKI -**Enables applications to use a smart card to digitally sign data and manage user certificates.
- **SATSA-CRYPTO -** A general-purpose cryptographic API that supports message digests, digital signatures, and ciphers.

The Sun Java™ Wireless Toolkit for CLDC emulator fully supports SATSA. This chapter describes how you can use the Sun Java™ Wireless Toolkit for CLDC to work with SATSA in your own applications.

For a more general introduction to SATSA and using smart cards with small devices, see the *SATSA Developer's Guide*, which is available at `http://java.sun.com/j2me/docs/satsa-dg/`.

The Sun Java™ Wireless Toolkit for CLDC includes the Java Card Platform Simulator, which you can use to simulate smart cards in the Sun Java™ Wireless Toolkit for CLDC emulator's slots. The Java Card Platform Simulator is found in.

Windows:	*toolkit*`\bin\cref.exe`
Linux:	*toolkit*`/bin/cref`

Hereafter we refer to it as simply `cref`.

If you need to develop your own Java Card applications, download the Java Card Development Kit, available at `http://java.sun.com/products/javacard/`.

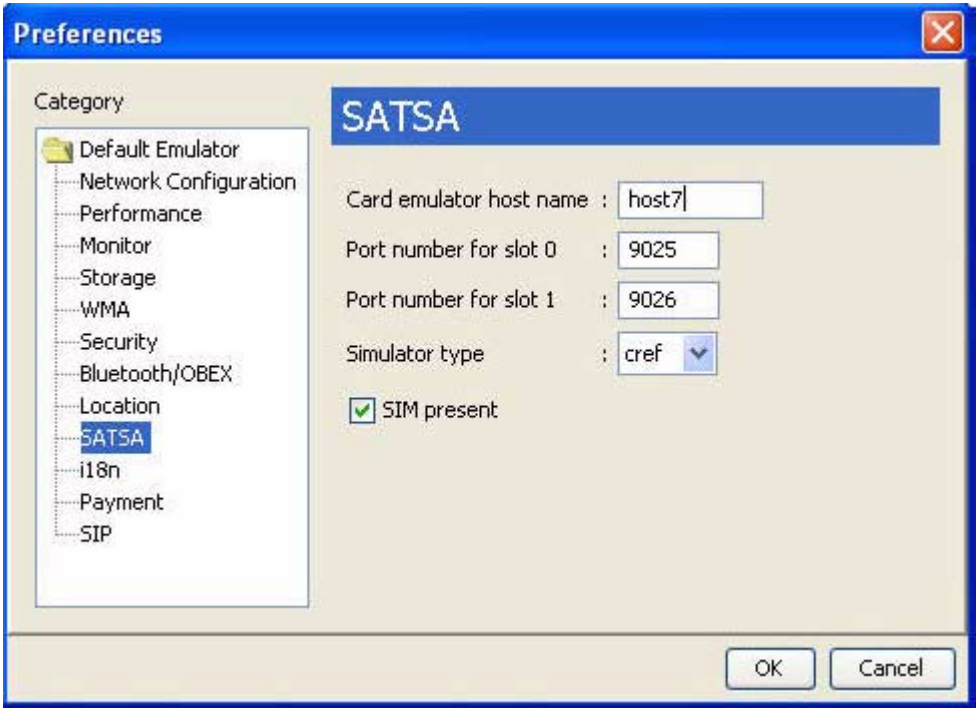
14.1 Card Slots in the Emulator

Real SATSA devices are likely to have one or more slots that house smart cards. Applications that use SATSA to communicate with smart cards need to specify a slot and a card application.

The Sun Java™ Wireless Toolkit for CLDC emulator is not a real device and, therefore, does not have physical slots for smart cards. Instead, it communicates with a smart card application using a socketprotocol. The other end of the socket might be a smart card simulator or it might be a proxy that talks with real smart card hardware.

The Sun Java™ Wireless Toolkit for CLDC emulator includes two simulated smart card slots. Each slot has an associated socket that represents one end of the protocol that is used to communicate with smart card applications. You can set the socket port number for each slot. Choose Edit > Preferences, then click the SATSA tab. The default ports are 9025 for slot 0 and 9026 for slot 1.

FIGURE 14-1 Setting Port Numbers for Smart Card Slots



14.2 Using the Java Card Platform Simulator

The basic procedure for testing SATSA applications with the Sun Java™ Wireless Toolkit for CLDC is as follows:

1. **Start `cref` with a Java Card platform application.**

2. **Start the Sun Java™ Wireless Toolkit for CLDC emulator.**

   When a SATSA application attempts to communicate with a smart card, it uses a socket connection to communicate with `cref`.

   It's important, therefore, to make sure that you start `cref` with the same port number as one of the slot port numbers you specified in the Sun Java™ Wireless Toolkit for CLDC preferences.

   For example, you could run `cref` on port 9025 with a prebuilt memory image using a command line like this:

   ```
   cref -p 9025 -i memory_image.eeprom
   ```

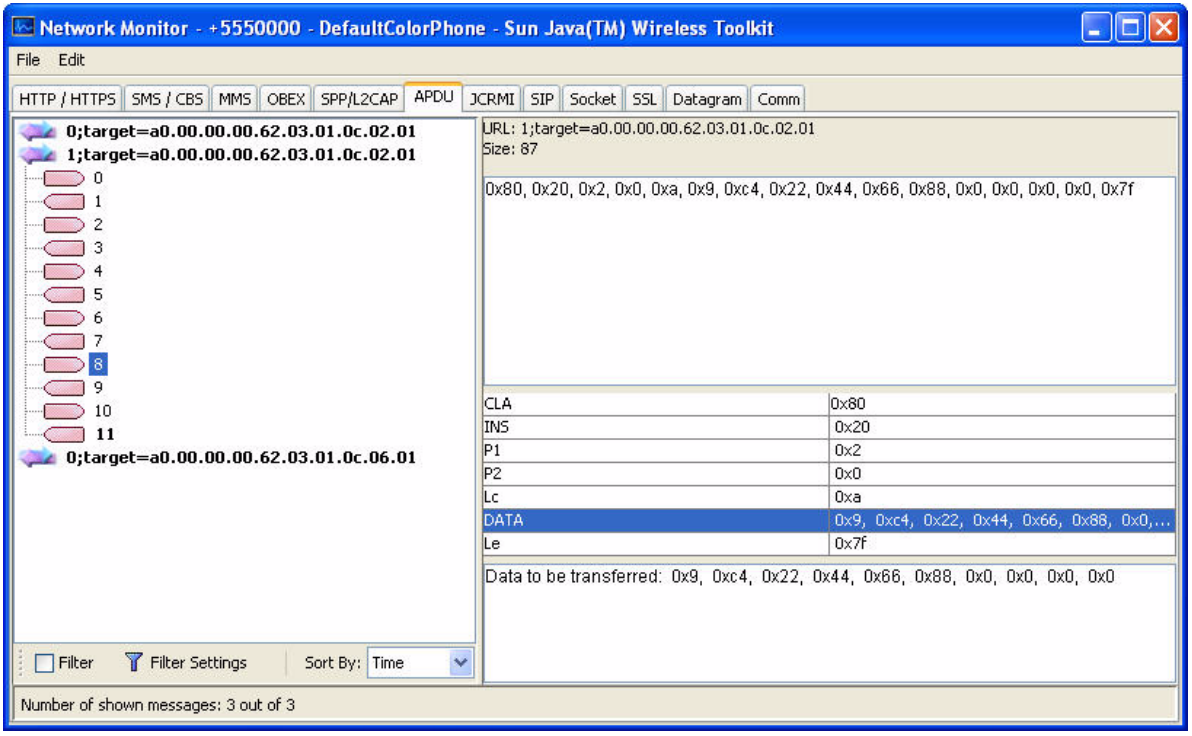
The Sun Java™ Wireless Toolkit for CLDC includes a demonstration application, `Mohair`, which illustrates how to use SATSA. For detailed instructions on running `Mohair`, see Appendix A.

14.3 Using the Network Monitor with SATSA

The Sun Java™ Wireless Toolkit for CLDC can display data exchanged with simulated smart card applications in the network monitor. The network monitor displays Application Protocol Data Units (APDUs) that are exchanged between the emulator and the smart card simulator. It can also show data exchanged using the Java Card Remote Method Invocation (Java Card RMI) protocol. The APDU and JCRMI tabs in the network monitor show data exchanged with a smart card.

The network monitor parses each APDU and shows fields in the request and response as appropriate.

FIGURE 14-2 Viewing an APDU in the Network Monitor



14.4 Adjusting Access Control

Access control permissions and PIN properties can be specified in text files. When the first APDU or Java Card RMI connection is established, the implementation reads the ACL and PIN data from the `acl_`*slot-number* in the *workdir*`\appdb` directory. For example, an access control file for slot 0 is *workdir*`\appdb\acl_0`. If the file is absent or contains errors, the access control verification for this slot is disabled.

The file can contain information about PIN properties and application permissions.

14.4.1 Specifying PIN Properties

PIN properties are represented by a `pin_data` record in the access control file.

```
pin_data {
    label string
    id number
    type        bcd | ascii | utf | half-nibble | iso
    min         minLength -
    stored      storedLength
    max         maxLength
    reference   byte
    pad         byte - optional
    flag        case-sensitive | change-disabled |
                unblock-disabled | needs-padding |
                disable-allowed | unblockingPIN
   }
```

14.4.2 Specifying Application Permissions

Application permissions are defined in access control file (`acf`) records.

```
acf AID fnumbers separated by blanks {
    ace {
         root CA name
         ...
         apdu {
                 eight numbers separated by blanks
                 ...
         }
```

```
            ...
            jcrmi {
                    classes {
                      classname
                      ...
                      }
                       hashModifier string
                       methods {
                      method name and signatiure
                      ...
                    }
            }
            ...
            pin_apdu {
                    id number
               verify | change | disable | enable | unblock
               four hexadecimal numbers
                    ...
            }
            ...
            pin_jcrmi {
                   id number
               verify | change | disable | enable | unblock
               method name and signature
                    ...
              }
           ...
           }
      ...
}
```

The `acf` record is an Access Control File. The AID after `acf` identifies the application. A missing AID indicates that the entry applies to all applications. The `acf` record can contain `ace` records. If there are no `ace` records, access to an application is restricted by this `acf`.

The `ace` record is an Access Control Entry. It can contain `root`, `apdu`, `jcrmi`, `pin_apdu`, and `pin_jcrmi` records.

The `root` record contains one CA name. If the MIDlet suite was authorized using a certificate issued by this CA, this `ace` grants access to this MIDlet. A missing `root` field indicates that the `ace` applies to all identified parties. One principal is described by one line. This line must contain only the word `root` and the principal name, for example:

```
root CN=thehost;OU=JCT;O=dummy CA;L=Santa Clara;ST=CA;C=US
```

The `apdu` or `jcrmi` record describes an APDU or Java Card RMI permission. A missing permission record indicates that all operations are allowed.

An APDU permission contains one or more sequences of eight hexadecimal values, separated by blanks. The first four bytes describe the APDU command and the other four bytes are the mask, for example:

```
apdu {
    0 20  0 82  0 20  0 82
    80 20  0  0 ff ff  0  0
}
```

The Java Card RMI permission contains information about the hash modifier (optional), class list, and method list (optional). If the list of methods is empty, an application is allowed to invoke all the remote methods of interfaces in the list of classes, for example:

```
jcrmi {
    classes {
             com.sun.javacard.samples.RMIDemo.Purse
    }
    hashModifier zzz
    methods {
        debit(S)V
        setAccountNumber([B)V
        getAccountNumber()[B
    }
}
```

All the numbers are hexadecimal. Tabulation, blank, CR, and LF symbols are used as separators. Separators can be omitted before and after symbols `{` and `}`.

The `pin_apdu` and `pin_jcrmi` records contain information necessary for PIN entry methods, which is the PIN identifier and APDU command headers, or remote method names.

14.4.3 Access Control File Example

```
pin_data {
  label   Unblock pin
  id        44
  type      utf
  min       4
  stored    8
  max       8
  reference 33
  pad       ff
  flag      needs-padding
```

```
  yflag     unblockingPIN
}
pin_data {
  label     Main pin
  id        55
  type      half-nibble
  min       4
  stored    8
  max       8
  reference 12
  pad       ff
  flag      disable-allowed
  flag      needs-padding
}

acf a0 0 0 0 62 ff 1 {
  ace {
        root CN=thehost;OU=JCT;O=dummy CA;L=Santa Clara;ST=CA;C=US

        pin_jcrmi {
            id 55
            verify enterPIN([B)S
            change changePIN([B[B)S
            disable disablePIN([B)S
            enable enablePIN([B)S
            unblock unblockPIN([B[B)S
           }
  }
}

acf a0 0 0 0 62 ee 1 {
  ace {
    root CN=thehost;OU=JCT;O=dummy CA;L=Santa Clara;ST=CA;C=US

    pin_apdu {
            id 55
            verify 1 2 3 1
            change 4 3 2 2
            disable 1 1 1 3
            enable 5 5 5 4
            unblock 7 7 7 5
        }
  }
}

acf a0 0 0 0 62 3 1 c 8 1 {
  ace {
        root CN=thehost;OU=JCT;O=dummy CA;L=Santa Clara;ST=CA;C=US
```

```
        jcrmi {
                  classes {
                      com.sun.javacard.samples.RMIDemo.Purse
               }
                  hashModifier xxx
                  methods {
                    setAccountNumber([B)V
                    getBalance()S
                    credit(S)V
                  }
  }
}
  ace {
        jcrmi {
                 classes {
                      com.sun.javacard.samples.RMIDemo.Purse
                   }

                    debit(S)V
                    getAccountNumber()[B
               }
             }
       }
  }

acf a0 00 00 00 62 03 01 0c 02 01 {
  ace {
        root CN=thehost;OU=JCT;O=dummy CA;L=Santa Clara;ST=CA;C=US
        apdu {
               0 20  0 82  0 20  0 82
               80 20  0  0 ff ff  0  0
             }
        apdu {
               80 22  0  0 ff ff  0  0
             }
        }
  }
acf a0 00 00 00 62 03 01 0c 02 01 {

  ace {
    apdu {
        0 20 0 82 ff ff ff ff
    }
  }
}

acf a0 00 00 00 62 03 01 0c 06 01 {

  ace {
```

```
    apdu {
        0 20 0 82 ff ff ff ff
    }
  }
}
```

CHAPTER 15

Using SIP

The Sun Java™ Wireless Toolkit for CLDC supports the SIP API for J2ME (JSR 180) with a proxy server, registrar, and network monitor support.

Session Initiation Protocol (SIP) is defined by RFC 3261, available at http://www.ietf.org/rfc/rfc3261.txt.

SIP provides a standard way for applications to set up communications. The application determines what communication actually takes place. SIP can be used to set up instant messaging, text chat, voice chat, video conferencing, or other types of sessions.

15.1 Understanding the Registrar and Proxy

A SIP registrar enables client applications to associate a user name with a specific network address. In essence, registering provides a way for a user to say "Here I am!"

A SIP proxy server is really just an entry point into a larger network of proxy servers. SIP messages that arrive at one proxy are routed to an appropriate destination, which is usually another proxy server or an end point, like a desktop computer or a mobile device. Although SIP messages can be sent directly between devices, they are usually routed through a proxy server.

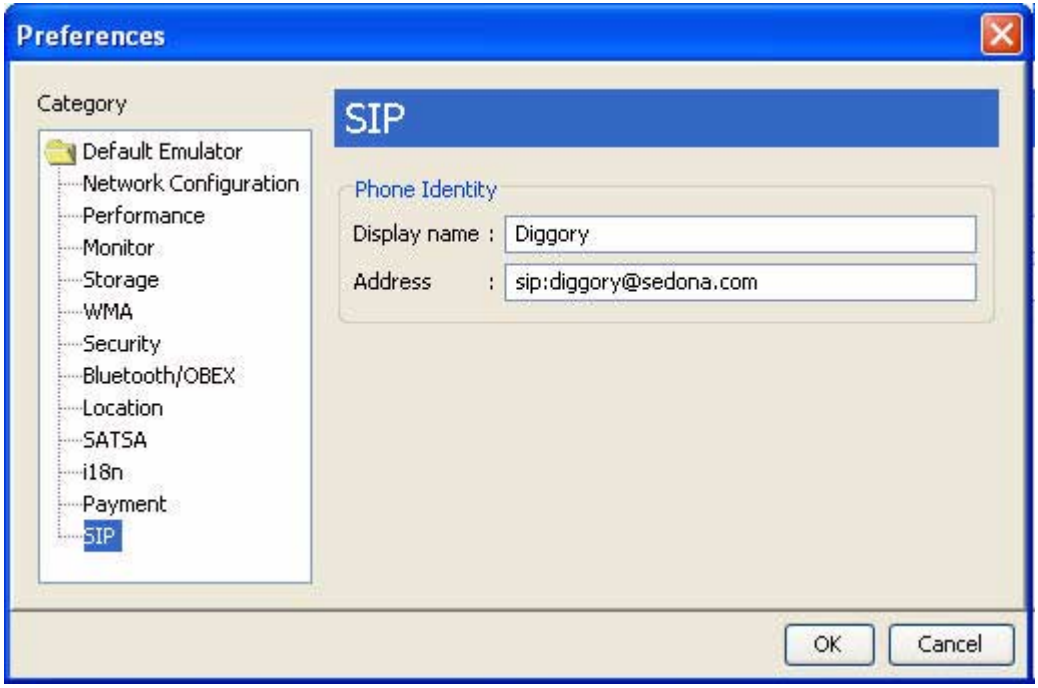
For example, suppose Diggory wants to start a video conference with Polly. Polly is on the road and her mobile phone sends a message to a registrar that associates her name with the mobile phone's network address. When Diggory tries to set up the video conference with Polly, his application uses SIP to ask the registrar for Polly's current network location.

The Sun Java™ Wireless Toolkit for CLDC includes a very simple SIP proxy and registrar server that you can use for testing applications that use the SIP API. You can also configure the toolkit to use an external proxy server and registrar server.

15.2 SIP Settings

To adjust settings for the Sun Java™ Wireless Toolkit for CLDC's SIP environment, choose Edit > Preferences and click SIP.

FIGURE 15-1 SIP Settings



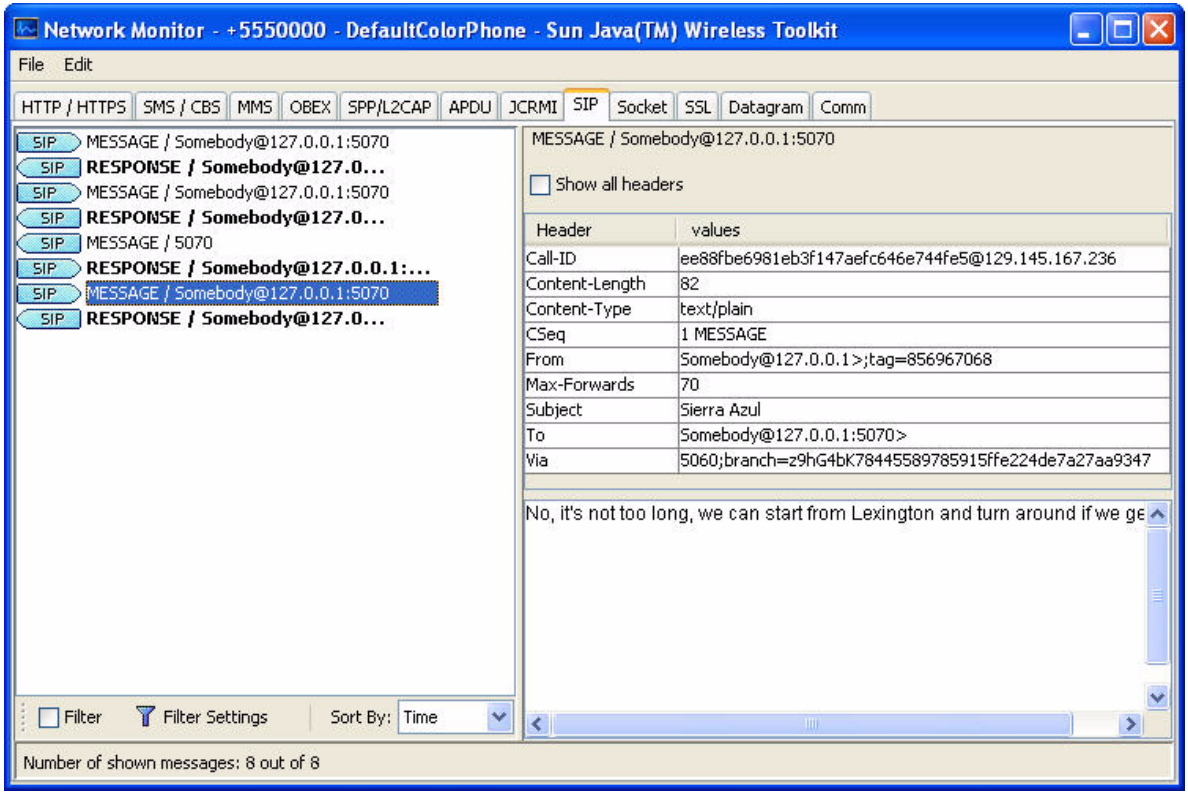
Display name and Address fields set the system properties `sip.display.name` and `sip.address`, respectively. Applications can use these system properties as a standard way to retrieve the identity associated with the device.

15.3 SIP Traffic in the Network Monitor

Network data that is sent and received using the SIP API can be recorded with the network monitor. The network monitor is fully described in Chapter 5. You can use the network monitor to track SIP messages that are sent to or from the emulator.

Click on the SIP tab to see SIP messages. SIP messages are shown in the left pane of the network monitor. Click a message to see its details in the right pane.

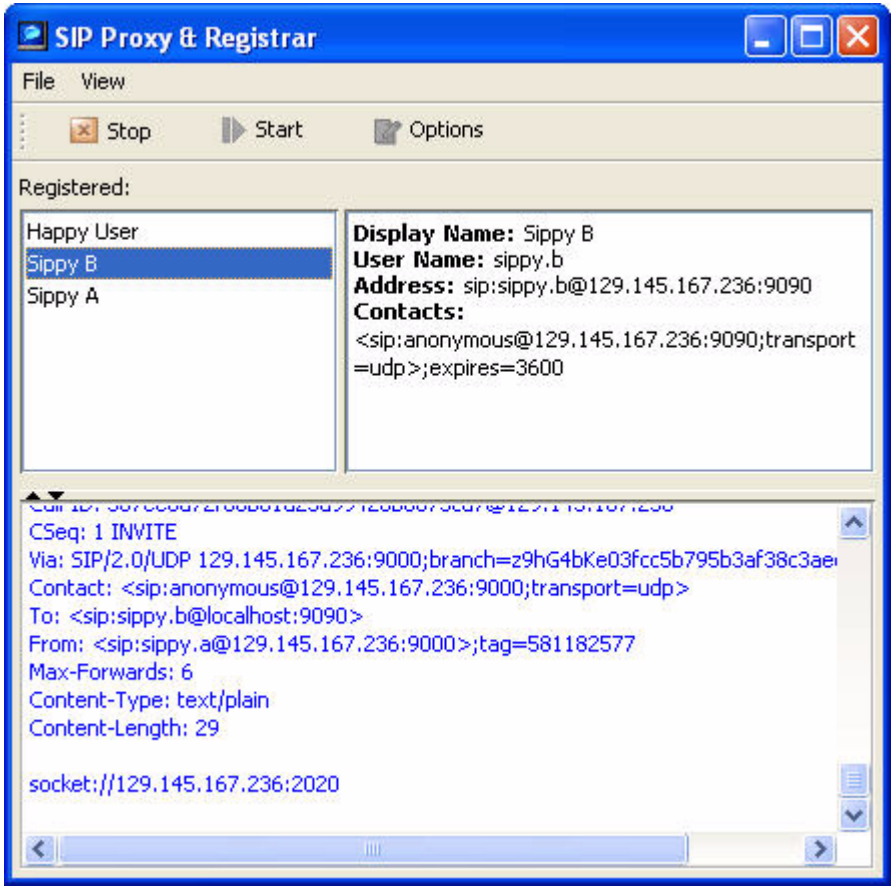
FIGURE 15-2 SIP Messages in the Network Monitor



15.4 SIP Proxy Server and Registrar

The Sun Java™ Wireless Toolkit for CLDC provides a simple SIP proxy server and registrar to make it easier to create SIP applications. To start the server, choose File > Utilities. Select Start SIP Server from the list and press Launch. The SIP server console window appears.

FIGURE 15-3 SIP Server Console

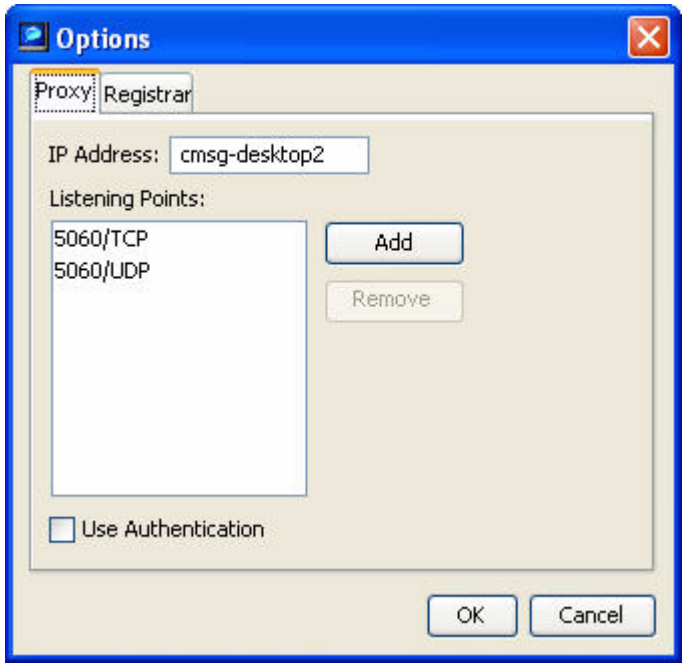


To stop the server, click Stop. To start the server, click Start.

While the server is running, the top left pane shows all users known to the registrar. Click a user name to see details about the user in the top right pane. The bottom pane of the window is a console that shows SIP messages that are received and sent by the proxy.

You can adjust the server options when the server is not running. Stop the server and click Options to see the options window (see FIGURE 15-4).

FIGURE 15-4 Setting Proxy Options



On the Proxy tab, you can set the IP address and ports upon which the server listens for incoming messages. Note that 5060, the default listening point, is a well known port for the SIP proxy. If you are working in a multiuser environment there is a chance another user may be using the port and you might accidentally connect to someone else's SIP server instance(the SIP server does not have any authentication mechanism and TCP/IP ports are freely accessible). If this happens you must specify another port.

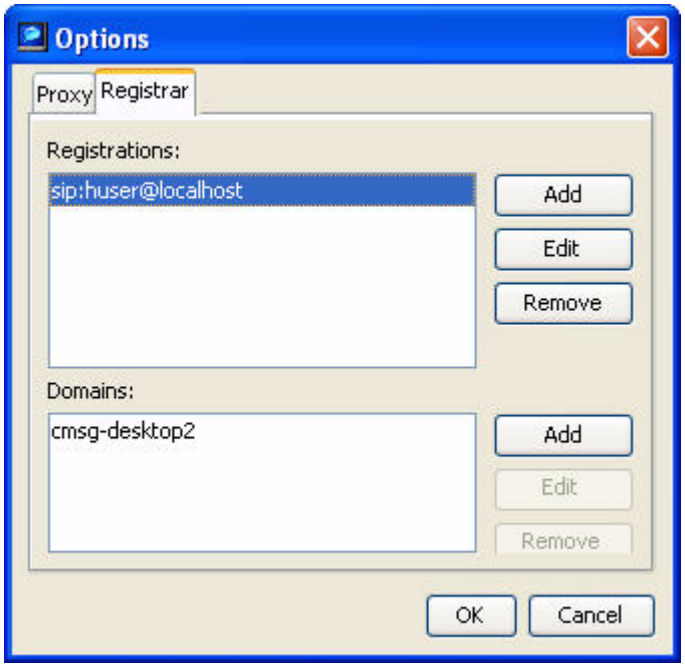
Click Add to specify more ports and their types. Select a port and click Remove to remove a listening port.

Check Use Authentication to force connecting clients to authenticate themselves to the server. The scheme used is digest authentication, which is described in section 22.4 of RFC 3261. SIP's digest authentication is nearly identical to HTTP digest authentication.

On the Registrar tab, you can set up the users and domains known to the registrar. The top list contains SIP users that are automatically registered when the SIP server is started. You can add a new user, edit an existing user, or remove a user.

In addition, you can adjust the list of domains managed by this registrar. Press Add to add a domain, Edit to edit an existing domain name, or Remove to remove a domain.

FIGURE 15-5 Setting Registrar Options



CHAPTER 16

Working with the Payment API

JSR 229, the Payment API, enables applications to make payments on behalf of their users. The Payment API supports different payment mechanisms through payment *adapters*. A device that implements the Payment API has one or more adapters. MIDlet suites use descriptor attributes to specify what types of payment adapters they can use.

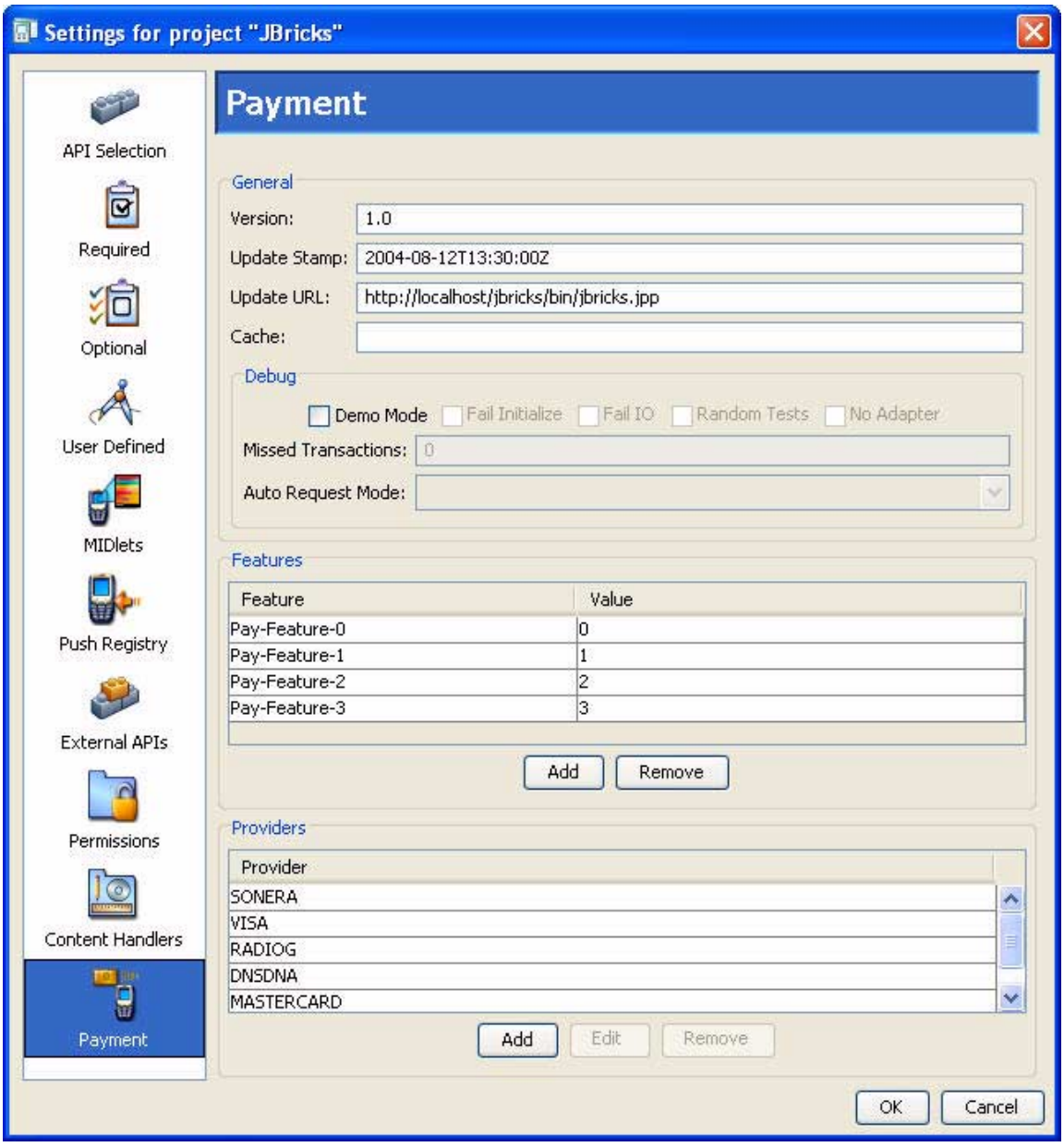
The Sun Java™ Wireless Toolkit for CLDC's emulator implements the Payment API with an example payment adapter that simulates both Premium Priced SMS (PPSMS) and credit card payments. In addition, the toolkit makes it easy to set the necessary attributes in the MIDlet's descriptor and JAR file manifest. Finally, a payment console enables you to easily track payments made or attempted by an application.

Because the Payment API is closely tied to provisioning and external device payment mechanisms, and because payments can only succeed in a trusted protection domain, always test and debug your Payment API applications using the toolkit's Run via OTA mechanism. See Chapter 2 for details.

16.1 Project Settings for Payment

To adjust the payment attributes for a project, click Settings and select the Payment icon.

FIGURE 16-1 Payment Settings



The fields and values are explained fully in JSR 229, the Payment API specification.

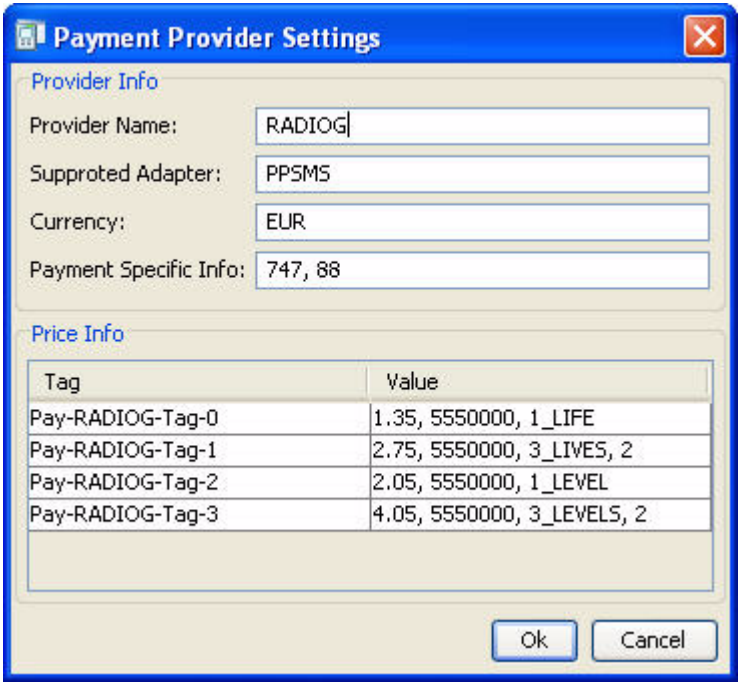
The General box contains information about the Payment API version in use and where to find payment updates. For testing, you can specify a `localhost` URL (as shown in the screen shot) that retrieves an update file directly from your project directory.

The Debug box contains options that are useful during application testing. Each option is explained in the Payment API specification.

The Features box lists the features your application can charge. These features correspond to the pricing information that is listed for each provider. You can modify the list of features by using the Add and Remove buttons.

The Providers box lists specific payment providers that can be used for this application. When the time comes to make a payment, the emulator (or device) matches one of its available payment adapters to one of the providers listed for the application. You can modify the list of providers with the Add, Edit, and Remove buttons. If you add or edit a provider, the following window appears.

FIGURE 16-2 Editing a Payment Provider



These fields are also described fully in the Payment API specification.

The Price Info box contains one line for each defined payment feature. To edit a value for a price tag, double click the corresponding cell in the Value column.

16.2 Editing Payment Attributes Directly

Payment attributes are stored in a *payment update file* with a `.jpp` extension. Read the specification for full details. The Sun Java™ Wireless Toolkit for CLDC provides a utility that makes it easy to edit the payment update file independently of the project settings.

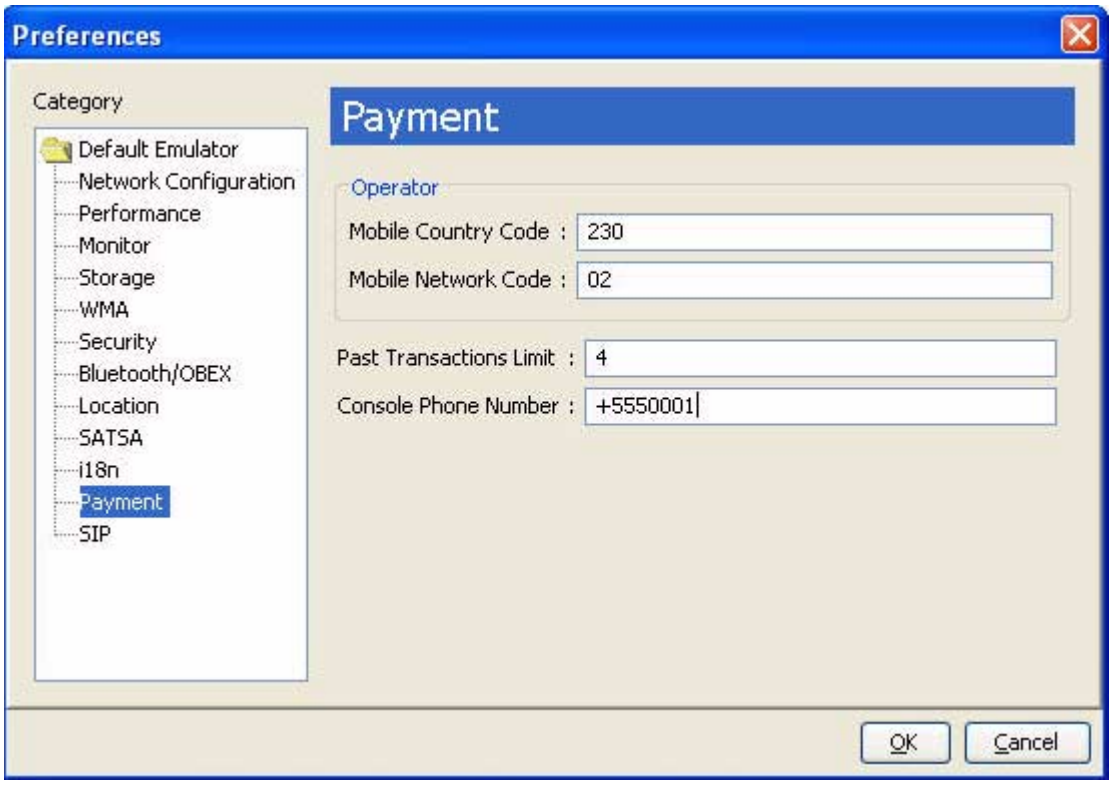
To run the utility, choose File > Utilities, select Payment Edit Dialog, and press Launch. You are prompted to select the payment update file you wish to edit. After you choose a file, a window that looks nearly identical to the Payment section of the project settings appears. The debug settings are not included in the payment edit utility.

Edit the payment update settings and press OK when you're finished. Because payment update files are cryptographically signed, the toolkit shows you a list of keys that you can use to sign the file. Select the key you wish to use and press Sign Payment Update File.

16.3 Payment Preferences

To adjust the Payment API settings for the toolkit, chooseEdit > Preferences and click Payment.

FIGURE 16-3 Setting Payment Preferences



The Mobile Country Code (MCC) and Mobile Network Code (MNC) are used in conjunction with PPSMS payment providers. Taken together, the MCC and MNC identify the wireless carrier of a device. At payment time, the MCC and MNC are used to find a matching provider from the list of providers in the project settings. Because the emulator is not a real device, you can simulate a carrier by filling in appropriate values for MCC and MNC. Consult the Payment API specification for more details.

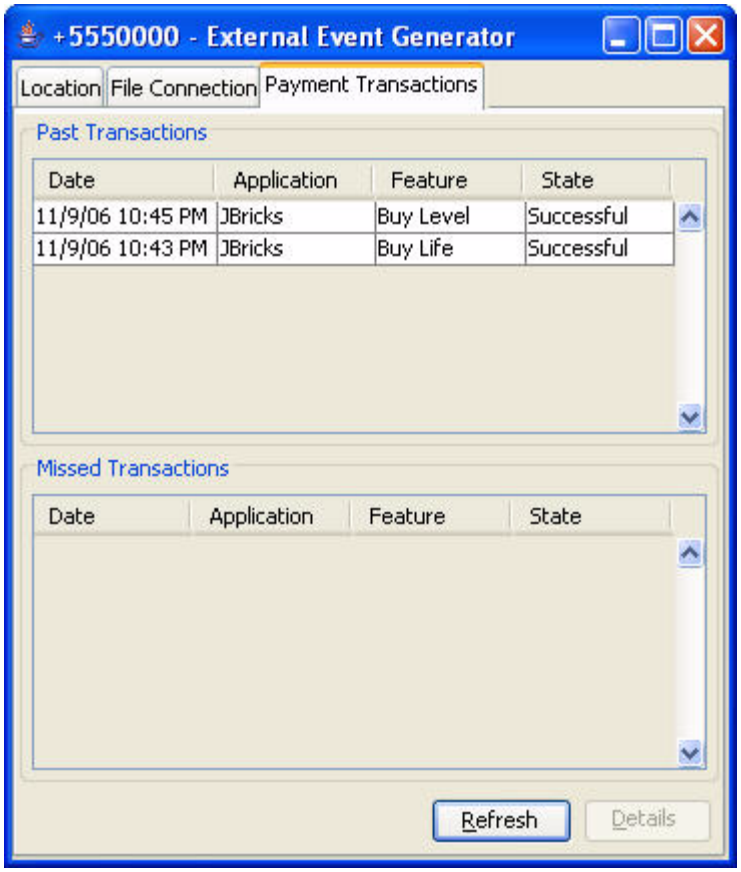
Past Transactions Limit determines how many previous transactions are recorded in the emulator. This affects the length of the list shown in the external event window, described below, as well as the number of past transactions that can be retrieved by the application itself.

Finally, Console Phone Number determines the simulated phone number of the payment console, which is described later.

16.4 Viewing Transaction History

The emulator keeps track of payment transactions, just as a real device does. To see the history of transactions, choose MIDlet > External Events from the emulator window's menu. Click the Payment Transactions tab.

FIGURE 16-4 Viewing Payment Transactions



Click Refresh to update the list after making more payments. Select a transaction and click Details to see all the details in a separate window.

The external events window only shows transactions that have been made while the emulator is running via OTA. Although it is possible to complete transactions without using OTA, such transactions are not shown. For the most realistic simulation of payments, always use Run via OTA to test applications.

16.5 Monitoring Payments

The Sun Java™ Wireless Toolkit for CLDC provides a payment console that makes it easy to see payments passing through the example payment adapter. To launch the Payment Service Provider Console, select File > Utilities > Payment Console.

FIGURE 16-5 Payment Console

In addition, you can view transactions using the network monitor. Credit card transactions are conducted using HTTPS, while PPSMS transactions use SMS. For a full description of the network monitor, see Chapter 5.

CHAPTER 17

Using the Mobile Internationalization API

JSR 238, the Mobile Internationalization API, is designed for applications that are to be displayed in multiple languages and used in multiple countries. The combination of country (or region) and language is a *locale*.

The central concept of JSR 238 is a *resource*, which is a string, image, or other object that is suitable for a particular locale. For example, an application that is to be distributed in Europe might include resources for Italian-speaking people living in Italy, Italian-speaking people living in Switzerland, Spanish-speaking people living in Spain, Spanish-speaking people living in Portugal, and so on.

Resources are stored in files in a format defined in JSR 238. The resource files are bundled as part of the MIDlet suite JAR file. The Sun Java™ Wireless Toolkit for CLDC provides a resource manager that simplifies the job of creating and maintaining resource files.

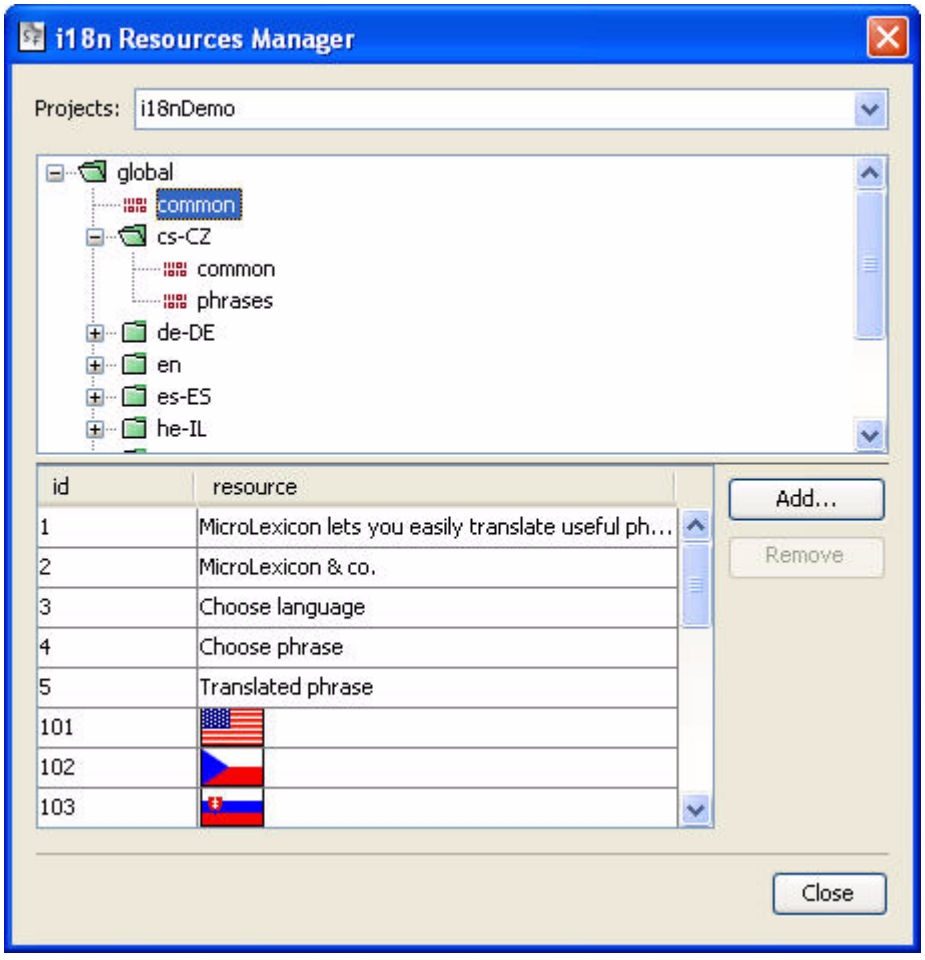
17.1 Setting the Emulator's Locale

A device's locale is contained in the system property `microedition.locale`. You can change the emulator's locale by choosing Edit > Preferences, then selecting i18n. Choose a locale from the combo box or type it in directly.

17.2 Viewing Application Resources

To launch the resource manager, choose File > Utilities. Select i18n Resources Manager and click Launch.

FIGURE 17-1 Resource Manager



First, choose your project from the Projects menu. All the resources for the selected project are shown in the rest of the window. If this is your first time looking at the resource manager, look at the resources for the `i18nDemo` demonstration application, which contains lots of interesting resources.

You can click + symbols to expand directories or - symbols to collapse them.

Nearly all other operations are available by right-clicking directories or resource files.

The top pane of the window shows the hierarchy of resource files in the application, while the bottom pane shows the contents of a resource file. In FIGURE 17-1, the contents of the top-level `common` resource file are being displayed.

17.3 Working With Locales

Locales are represented by directories under the top-level `global` directory. The locale directories contain resource files which, in turn, hold the actual resources that can be used by the application.

Locales are represented by standard language and country codes as described in the MIDP 2.0 specification. For example, `pt-BR` represents Portuguese-speaking people living in Brazil.

To add a locale directory, right-click on the top-level `global` directory and choose Add Locale. Choose the locale from the combo box, or type it directly, and click OK.

To rename a locale, right-click the locale directory and choose Rename.

To remove a locale and all its contained resource files, right-click the locale directory and choose Delete.

17.4 Working With Resource Files

Resource files can be global (at the top level) or inside a locale directory. To create a new global resource file, right-click the top-level `global` directory and choose Add new resource file. Choose a name for the file.

Rename a resource file by right-clicking the file and choosing Rename.

Remove a resource file by right-clicking the file and choosing Delete.

You can copy, cut, and paste entire resource files. Right-click a file and choose Copy or Cut. Then right-click the locale directory (or the top-level `global`) and choose Paste.

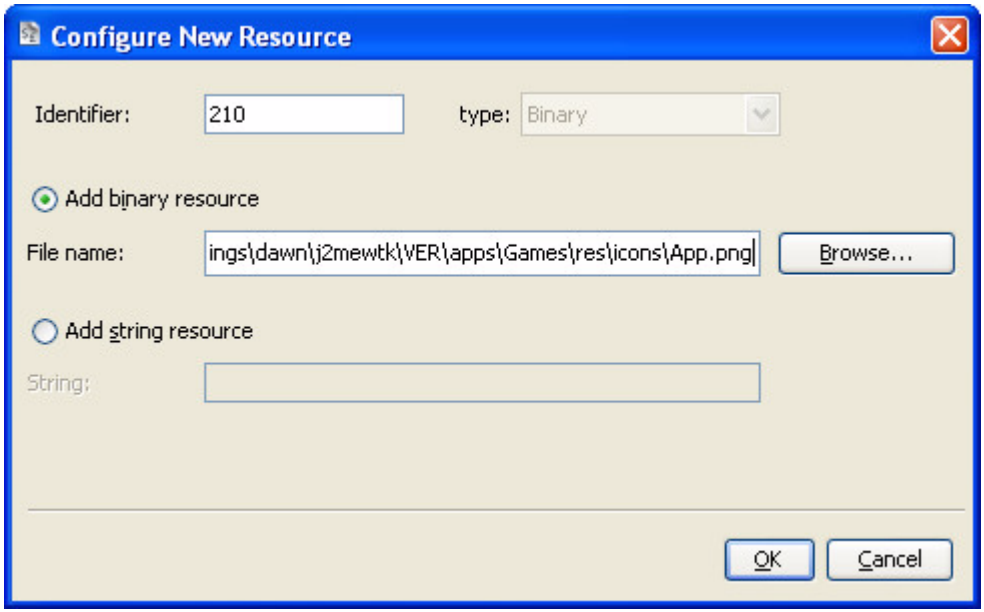
17.5 Working With Resources

Click on a resource file in the top pane of the resource manager window to see its contents in the bottom pane.

To add a new text resource, click on Add and select Add string resource. An identifier is automatically assigned, but you can enter a different one if you want. Click OK.

To add an image or another type of data, click Add and select Add binary resource. You can browse to whatever file you want to add. Again, you can change the identifier if you wish. Click OK to add the resource.

FIGURE 17-2 Adding an Image Resource



To edit a resource, double-click it. You can type a new value for a string resource. If you double-click file resources, you can choose another file.

APPENDIX **A**

Application Demonstrations

This appendix describes the application demonstrations that are bundled with the Sun Java™ Wireless Toolkit for CLDC.

A.1 Overview

The Sun Java™ Wireless Toolkit for CLDC includes demonstration applications that highlight some of the technologies and APIs that are supported by the emulator. The goal of these demonstrations is to ive you a glimpse of the API features of the emulator and the enhancements throughout the toolkit.

TABLE A-1 lists all the demonstration applications that are included in this release.

Most demonstration applications are simple to run. Section A.2, "General Instructions" on page A-4 contains instructions for running most demonstrations. Demonstrations that have additional documentation are linked in TABLE A-1. If there is no link, the demonstration is simple (or has its own instructions) and the general instructions are sufficient.

The source code for every demonstration application is available in *toolkit*`/apps` directory. Subdirectories contain projects, and each project has a `src` directory that contains Java programming language source code. For example, on Windows, if the toolkit is installed in `C:\WTK2.5.2`, the source code for the SMS sender MIDlet (`example.sms.SMSSend`) in `WMADemo` resides in `C:\WTK2.5.2\apps\WMADemo\`

`src\example\sms\SMSSend.java`. As discussed in Section 1.1.2, "Working Directory Files" on page 1-2, when you open a project it is copied to your *workdir*`/apps` directory.

TABLE A-1 Application Demonstrations

Demonstration	APIs	Description	Special Instructions
AdvancedMultimedia Supplements	JSR 234 Advanced Multimedia Supplements	Shows 3D audio, reverberation, image processing, and camera control.	A.3
Audiodemo	MMAPI 1.1	Demonstrates audio capabilities, including mixing and playing audio with an animation.	
BluetoothDemo	JSR 82 Bluetooth	Demonstrates device discovery and data exchange using Bluetooth.	A.4
CHAPIDemo	JSR 211 CHAPI	A content viewer that also makes uses of MediaHandler.	A.5
CityGuide	JSR 179 Location API	Shows a city map that displays landmarks based on the current location.	A.6
Demos	MIDP 2.0	Includes various examples: animation, color, networking, finance, and others.	A.7
Demo3D	JSR 184 Mobile 3D Graphics	Contains MIDlets that demonstrate how to use 3D graphics, both immediate mode and retained mode.	A.8
FPDemo	CLDC 1.1	Simple floating point calculator.	
Games	MIDP 2.0	Includes TilePuzzle, WormGame, and PushPuzzle.	
GoSIP	JSR 180 SIP	Demonstrates setting up a chat using SIP and a SIP server	A.9
i18nDemo	JSR 238 Mobile Internationalization API	Includes string sorting, number formatting, and a phrase translator.	A.10
JBricks	JSR 229 Payment API	A game that uses the Payment API for buying extra lives or levels.	A.11

TABLE A-1 Application Demonstrations (Continued)

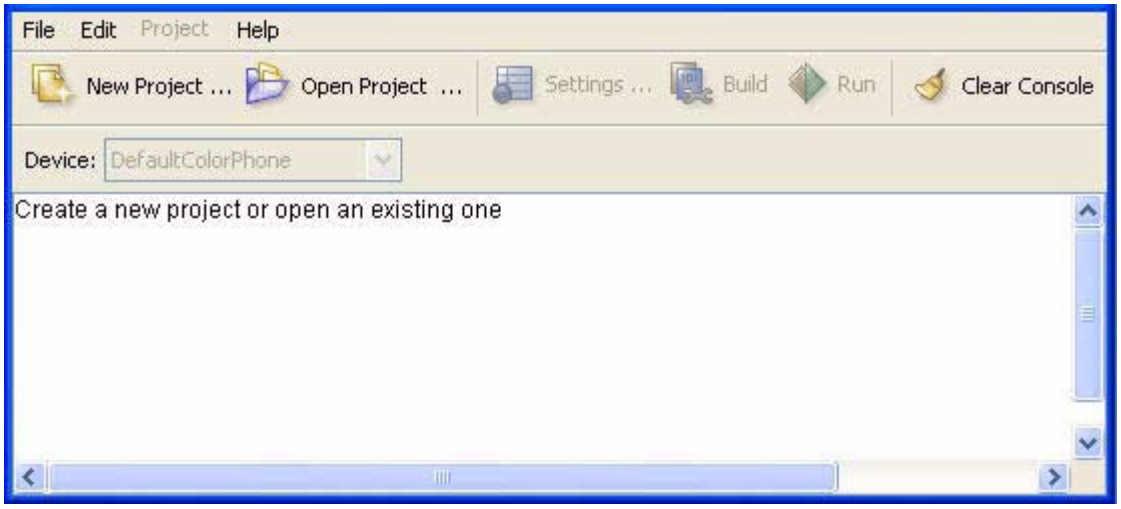
Demonstration	APIs	Description	Special Instructions
JSR172Demo	Web services	Demonstrates how to use the JSR 172 API to connect to a web service from a MIDlet.	A.12
MobileMediaAPI	MMAPI 1.1	Demonstrates MMAPI features, including tone sequences, MIDI playback, sampled audio playback, and video.	A.13
NetworkDemo	MIDP 2.0	Shows how to use datagrams and serial connections.	A.14
ObexDemo	JSR 82 OBEX	Demonstrates transferring data using OBEX over IrDA.	A.15
OpenGL® ES Demo	JSR 239 OpenGL® ES	Shows how to create 3D graphics using OpenGL® ES.	
PDAPDemo	JSR 75 PIM and FileConnection	Shows how to manipulate contacts, calendar items, and to-do items. Demonstrates accessing local files.	A.16
Photoalbum	MIDP 2.0	Demonstrates a variety of image formats.	
SATSADemos	JSR 177 SATSA	Demonstrates communication with a smart card and other features of SATSA.	A.17
SATSAJCRMIDemo	JSR 177 SATSA	Shows how to use SATSA-Java Card RMI.	A.18
SIPDemo	JSR 180 SIP	Simple message exchange using SIP.	A.19
SnapMobileSample		Demonstrates the connected community game play feature.	
SVGContactList	JSR 226 SVG API	Demonstrates a contact list displayed with different skins.	A.20
SVGDemo	JSR 226 SVG API	Shows different techniques for rendering SVG.	A.21
UIDemo	MIDP 2.0	Showcases the breadth of MIDP 2.0's user interface capabilities	
WMADemo	WMA 2.0	Shows how to send and receive SMS, and CBS, and MMS messages.	A.22

A.2 General Instructions

Most of the demonstration applications can be run then launched with no special preparation. Some demonstrations, however, require changes to the toolkit preferences or settings. This section describes the general procedure.

The first step is to run the toolkit. To do this, go to the Microsoft Windows Start menu and choose Start > All Programs > Sun Java Wireless Toolkit 2.5.2 for CLDC > Wireless Toolkit 2.5.2. The user interface appears as shown in FIGURE A-1.

FIGURE A-1 Wireless Toolkit User Interface



Click the Open Project button to open a demonstration application. A list of all the available applications appears. As discussed in Section 1.1.2, "Working Directory Files" on page 1-2, projects in the installation directory are *italicized*, and projects in your working directory are shown in **bold** (FIGURE 1-1). When you open a project in the installation directory a copy is created in your working directory, then opened.

Once the application is open you can press the Run button in the toolbar, or if an installation on the emulator is required, click Project > Run via OTA

The device emulator window opens with the demo application running. If there is a menu of MIDlets, use the navigation arrows to choose an item, then choose SELECT. As the demonstration progresses you might need to press one of the soft keys below the screen on the left or right side. You use soft keys to istall or launch an application, open a menu, exit, or perform some other action. Some examples include these instructions.

Some demonstrations require specific setup and instructions. For example, if an example uses web services and you are behind a firewall, you must configure the emulator's proxy server settings or the demo will fail:

- Choose Edit > Preferences
- Select the Network Configuration icon.
- Check Use proxy server.
- Fill in the proxy server address field and the port number.

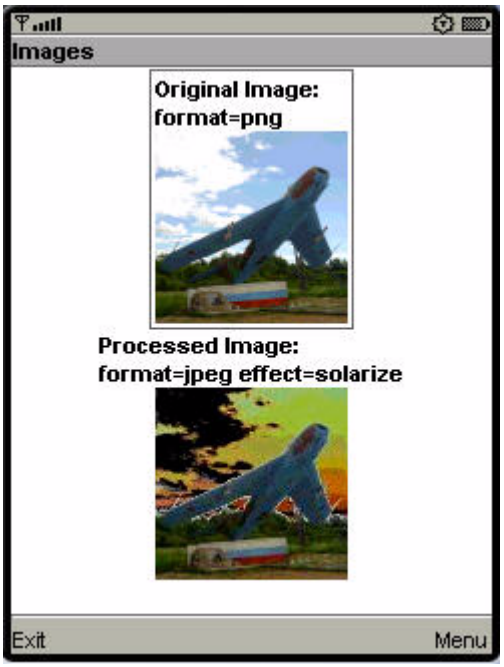
Read each demonstration description for more operating instructions.

A.3 Advanced Multimedia Supplements

This MIDlet suite demonstrates the power of JSR 234 Advanced Multimedia Supplements (AMMS). It consists of the following MIDlets:

- **Image Effects -** Shows standard image processing operations. Choose an effect from the menu. The processed image is shown following the source image. Note that some items, Set Transforms, for example, can perform several operations. Click the Done soft button to apply every effect.

FIGURE A-2 Processing Images in a MIDlet



- **Radio Tuner -** Simulates a radio tuner by reading audio files from the project directory via the toolkit's built-in HTTP server.
- **Camera -** Shows how the Advanced Multimedia Supplements provide control of a device's camera. The screen shows the viewfinder of the camera (simulated with a movie). You can use commands in the menu to change the camera settings and take and manage snapshots.
- **Moving Helicopter -** Simulates a helicopter flying around a stationary observer. Use headphones for best results. You can control many of the parameters of the simulation, including the size of the helicopter, whether the Doppler effect is used, and the spectator orientation (for example, standing straight or lying face down).
- **Music Effects -** Shows off the advanced audio capabilities of the Advanced Multimedia Supplements. As an audio file loops continuously, you can adjust the volume, pan, equalizer settings, reverberation, and chorus settings.

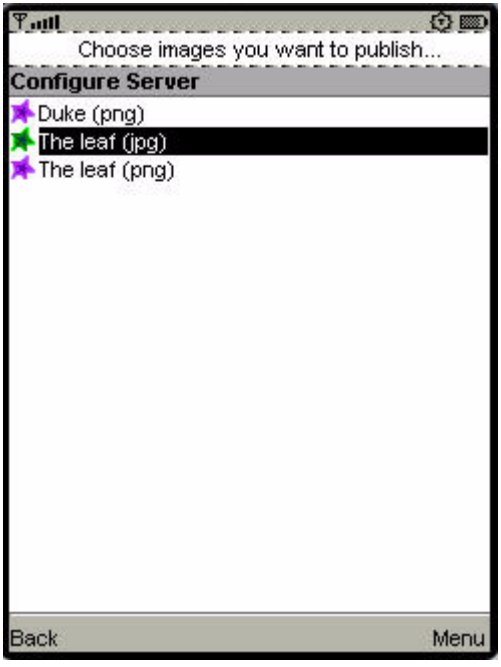
A.4 Bluetooth Demo

This application contains MIDlets that demonstrate the use of JSR 82's Bluetooth API.

The project BluetoothDemo shows how images can be transferred between devices using Bluetooth. You must run two instances of the emulator to see how this demonstration works.

In the first emulator, launch Bluetooth Demo, then choose Server. The emulator asks you if you want to allow a Bluetooth connection. Choose Yes. The server starts and displays a list of images. At the beginning, none of the images are available on the Bluetooth network. To make images available, select them and from the menu choose Publish image (or type or click 1). The icon color changes from purple to green, indicating it is published.

FIGURE A-3 Running the Bluetooth Demo Server



On the second emulator, launch Bluetooth Demo, then select Client. The MIDlet tells you it's ready to search for images. Click the Find soft button. The MIDlet finds the other emulator and get a list of images from it. Select one from the list and choose Load. The emulator asks if you want to allow the connection. Choose Yes.

- If you are running the demonstration in a trusted protection domain, the image is transferred using simulated Bluetooth and is shown on the client emulator.
- If you are not running in a trusted protection domain, the first emulator (the server) displays a prompt asking if you want to authorize the connection from the client. Choose Yes. The image is displayed in the client emulator.

FIGURE A-4 Image Transferred Using Simulated Bluetooth

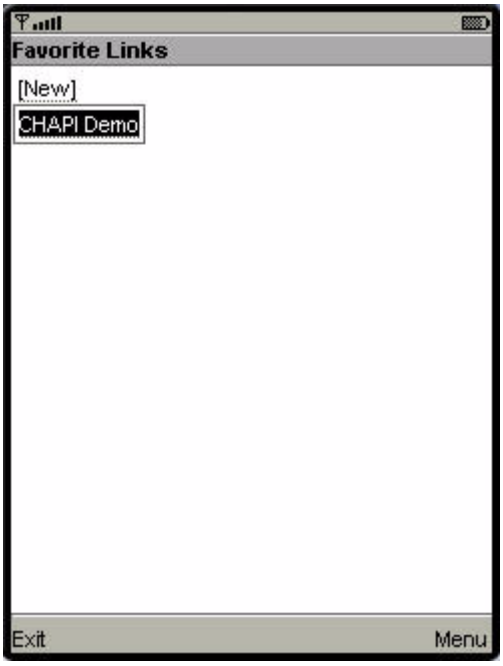
A.5 CHAPIDemo

CHAPIDemo is a content browser (see JSR 211). It maintains a list of favorites and enables you to select and view various kinds of content.

This demonstration uses the content handler registry, so you cannot see all of its features when you use the Run button. Instead, use Project > Run via OTA to install the application into the emulator. If you don't know how to do this, read about it in Section 2.3.2, "Install" on page 2-8.

After you install CHAPIDemo, it appears in the application list as Text Viewer. It is a MIDlet that is a content handler for plain text. Select Text Viewer and choose Launch from the soft button menu. A list of favorite links appears.

FIGURE A-5 Viewing Favorite Links in CHAPIDemo



Use the navigation keys to highlight CHAPIDemo then press SELECT on the emulator. The application asks if it is OK to use airtime. Press the Yes soft button. A list of various types of content appears (FIGURE A-6).

FIGURE A-6 Content List

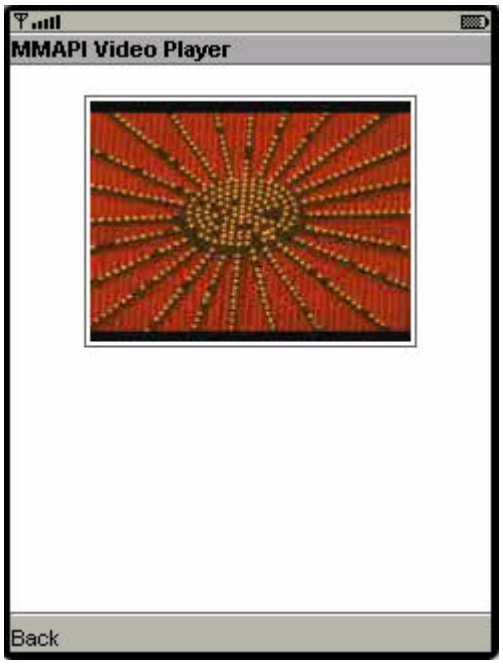
Try selecting one of the `Duke.png`. Use the arrows to highlight the link, then press SELECT to view the file. Using CHAPI, the `ImageViewer` MIDlet is launched and displays the content.

FIGURE A-7 Viewing a PNG Image

The other types of content require another content handler MIDlet suite, MediaHandler. To install this suite from CHAPIDemo, select the `MediaHandler.jad` link (the first item in the list shown in FIGURE A-6). The AMS is invoked and leads you through the installation.

After the MIDlet suite is installed, you can view the other types of content listed in Text Viewer. For example, select `http:video/test-mpeg.mpg` to see a series of images including the one shown in FIGURE A-8.

FIGURE A-8 Viewing an MPEG Movie Using MediaHandler



To view the content handler settings for the TextViewer and ImageViewer MIDlets, click Settings, then click on the Content Handlers icon. You might also wish to examine the MediaHandler project.

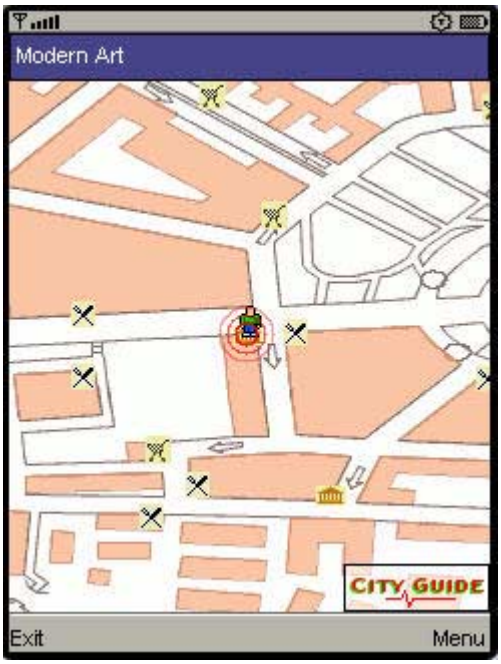
A.6 CityGuide

CityGuide demonstrates how to use the Location API (JSR 179). It shows a walker's current position superimposed on a city map. The walker moves around the city and landmarks are highlighted and identified as the walker approaches. In this demo we get the walker's location from an XML script named `citywalk.xml` (the event file) that submits the device location information. See Chapter 13 for a full explanation.

Because location prompts occur frequently, it is best to run this demonstration in manufacturer (trusted) mode, as explained in Section 6.2.1, "MSA Protection Domains" on page 6-3. In the user interface, select Edit > Preferences, then select Security. Choose Manufacturer for the Security domain.

Open and run the CityGuide project. In the emulator, launch the CityGuide MIDlet. Click Next to view the map page.

FIGURE A-9 Your Location in the City



Choose MIDlet > External events from the emulator window menu. On the Location tab click the browse button. Select the following event file in *workdir*`\apps\CityGuide\citywalk.xml`.

The player buttons at the bottom of the window are now active. See FIGURE 13-1. Press the green play button (right-pointing triangle) to run the script.

The display shows four types of landmarks: restaurants, museums, shops, and theaters. To adjust the landmark display, open the soft menu and choose the Settings command. See FIGURE A-10. Use the navigation keys to highlight a category, then use SELECT to check or uncheck an item.

When you are near a landmark (shown highlighted on the map), open the soft menu and choose the Detail command to see more information. See Chapter 13 for more details on location scripts.

FIGURE A-10 Location Settings

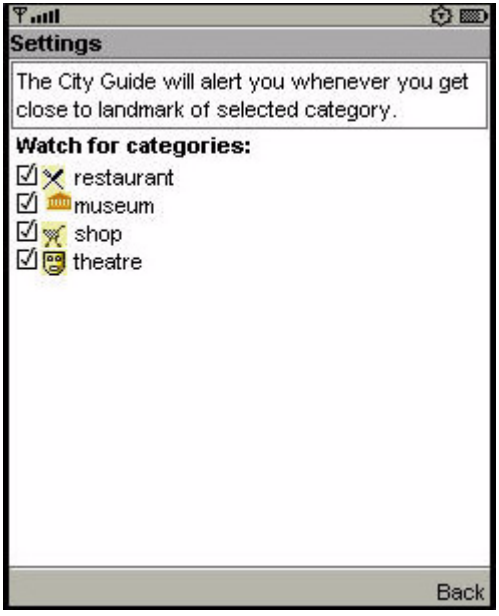
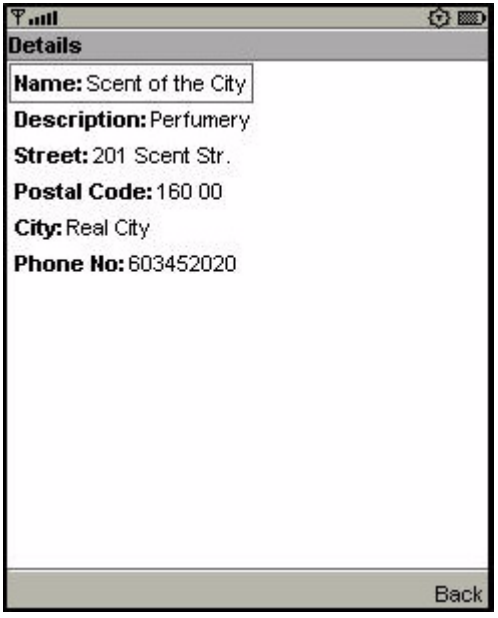


FIGURE A-11 Landmark Details



A.7 Demos

This demo contains several MIDlets that highlight different MIDP features.

A.7.1 Colors

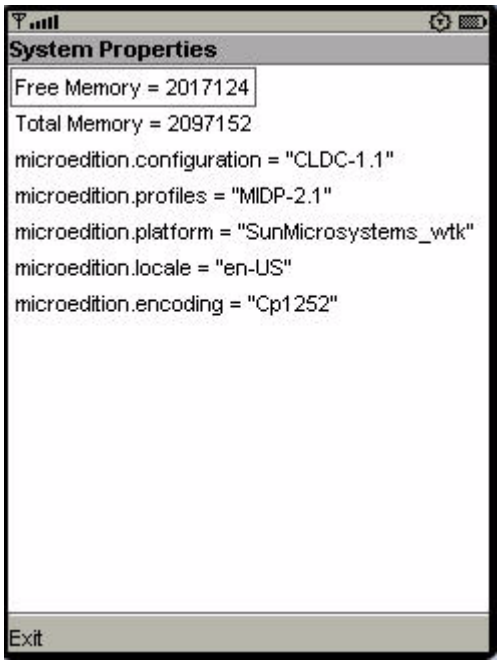
This application displays a large horizontal rectangle that runs the width of the screen. Below, ten small vertical rectangles span the screen. Finally, three horizontal color bars indicate values for blue, green, and red (RGB). Values are expressed as decimal (0-255) or hexadecimal (00-ff) based on the first menu selection.

- To select a vertical bar to change, use the up navigation arrow to move to the color bars. Use the right navigation arrow to highlight a color bar. The large rectangle becomes the color of the selected bar.
- Use the up or down selection arrows to choose the value to change (red, green, or blue). Use the left or right arrow keys to increase or decrease the selected value. The second menu item allows you to jump in increments of 4 (Fine) or 32 (coarse).
- You can change the color on any or all of the vertical bars.

A.7.2 Properties

This MIDlet displays property values. For example, see FIGURE A-12:

FIGURE A-12 System Properties



A.7.3 Http

This test application uses an HTTP connection to request a web page. The request is issued with HTTP protocol `GET` or `POST` methods. If the `HEAD` method is used, the head properties are read from the request.

Preparing to Run the Demo

Before beginning, examine your settings as follows.

- Go to Preferences > Security. Set the policy to JTWI and the domain to maximum.
- In Preferences > Network Configuration, the HTTP version must be 1.1.
- If you are behind a firewall, go to Preferences > Network Configuration and specify your proxy server information.
- If you are running antivirus software, you might need to create a rule that allows this MIDlet to allow connections to and from a specific web site.

Running the Demo

Launch the Http MIDlet. By default the MIDlet attempts to contact `http://www.yahoo.com`. To test, choose the Menu soft key and choose 1, 2, or 3 to test the selected URL.

Http Test returns the information it is able to obtain. If the information fills the screen use the down arrow to scroll to the end. The amount of information depends on the type of request and on the amount of `META` information the page provides. To provide body information or content, the page must declare `CONTENT-LENGTH` as described in RFC 2616.

Using Menu Options

Use the Menu soft key for the following actions.

- Choose 1 to `GET` information from the selected page.
- Choose 2 to obtain the `POST` information from the selected page.
- Choose 3 to display the `HEAD` attributes for the page.
- Choose 4 to bring up the current list of web pages. You can chose a new page or add your own page to the list. To specify a new URL, choose the Menu soft key and choose 4. The screen displays `http://`. Type in the rest of the URL, making sure to end with a slash (/). For example `http://www.internetnews.com/`. Press the OK soft button. The Http Test screen shows your new URL and prompts for an action.

A.7.4 FontTestlet

This MIDlet shows the various fonts available: Proportional, Regular, Regular Italic, Bold Plain, and Bold Italic. Choose 1 or 2 from the menu to toggle between the system font (sans serif) and the monospace font.

A.7.5 Stock

Like the Http demonstration, This sample uses an HTTP connection to obtain information. Use the same preparation steps as Section A.7.3, “Http” on page A-16.

Run the Demos project and launch the Stock MIDlet.

By default, the screen displays an empty ticker bar at the top. Below the ticker, the menu list shows four applications: Stock Tracker, What If? Alerts, and Settings. You must add stock symbols before you can use the first three applications.

A.7.5.1 Working with Settings

To use the applications features, you must supply some stock symbols for the application to act upon.

Add Stock Symbols to the Ticker

To add a stock symbol to the ticker, use the navigation arrows to select Settings.

Select Add Stock.

The display prompts you to enter a stock symbol. Type `SUNW` and select the Done soft key. The stock you added and its current value is now displayed in the ticker. Add a few more stock symbols, such as IBM and HPQ.

Change the Update Interval

By default the update interval is 15 minutes. Select Updates to change the interval. Use the navigation arrows to select one of Continuous, 15 minutes, 30 minutes, 1 hour, or 3 hours. Select the Done soft key.

Remove a Stock

Select Remove a Stock. You see a list of the stocks you have added. Use the navigation keys to select one or more stocks to remove. Choose the Done soft key.

A.7.5.2 Stock Tracker

Stock Tracker displays a list of the stocks you added and their current values. Stock tracker display additional information about the selected stock, for example, the last trade and the high and low values.

Choose a stock and press SELECT.

A.7.5.3 What If?

What If? is an application that asks for the original purchase price and the number of shares you own. It calculates your profit or loss based on the current price.

Select a stock symbol.

Enter the purchase price and the number of shares, then press Calc.

A.7.5.4 Alerts

This application sends you a notification when the price changes to a value you specify.

From the main menu, select Alerts.

Select Add.

Choose a Stock. The screen prompts, `Alert me when a stock reaches`. Enter an integer.

The alert is placed on the Current Alerts list. To remove an alert, press Remove and select the alert. Choose the Done soft key.

When the value is reached you will hear a ring and receive a message. For example, *Symbol* has reached your price point of $*value* and is currently trading at $*current_value*. Once the alert is triggered it disappears from the Current Alerts list.

A.7.6 Tickets

This demonstrates how an online ticket auction application might behave. The home screen displays a ticket ticker across the top. The Choose a Band field displays Alanis Morrisette by default.

To select a band, highlight the band name and press SELECT. Use the down arrow to highlight a different band, moby, for example, then press SELECT. The available auction appears.

To make a bid, select the Menu soft key and choose 2. Use the arrow keys to move from field to field. Fill out each field. Select the Next soft key. The application asks you to confirm your bid. Use the arrow keys to highlight Submit then press SELECT. You receive a Confirmation number. Click Bands to return to the welcome page.

To set an alert, select the Menu soft key and choose 3. Use the navigation arrows to move to the field and type in a value higher than the current bid. Select the Save soft key. You are returned to the welcome page. You can trigger the alert by making a bid that exceeds your alert value. Your settings determine how often the application checks for changes, so the alert may not sound for a few minutes.

To add a band, select the Menu soft key and choose 4. Type in a band name or a comma-separated list of names. Choose the Save soft key. After confirmation you are returned to the welcome page. The added band(s) are displayed in the Choose a Band drop down.

Note – This is only a demonstration. To fully describe the band you must edit the file *workdir*`\apps\Demos\src\example\auction\NewTicketAuction.java`.

To remove a band, select the Menu soft key and choose 5. Navigate to a band then choose SELECT to mark the check box. You can select multiple bands. Choose the Save soft key.

To display the current settings for ticker display, updates, alert volume, and date, select the Menu soft key and choose 6. If desired, use the arrow keys and the select key to change these values. Choose the Save soft key.

A.7.7 ManyBalls

This MIDlet starts with one ball traveling the screen. Use the up and down arrows to accelerate or decelerate the ball speed (fps). Use the right or left arrows to increase or decrease the number of balls.

A.8 Demo3D

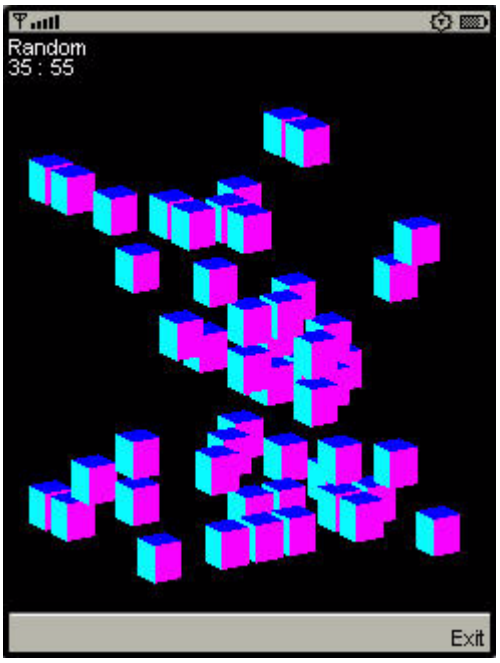
This application contains three MIDlets that show off the emulator's support of JSR 184, the Mobile 3D Graphics API.

A.8.1 Life3D

Life3D implements the popular Game of Life in three dimensions. Live cells are represented by cubes. Each cell has 26 possible neighbors (including diagonals). For each step of the animation, cells with fewer than four neighbors die of loneliness, while cells with more than five neighbors die of overcrowding. An empty cell with exactly four neighbors becomes a new live cell.

The view of the playing board rotates slowly so you can view the board from all angles.

FIGURE A-13 Game of Life in Three Dimensions



The keypad buttons in TABLE A-2 provide control over the game.

TABLE A-2 Controls for `Life3D`

Button	Description
0	Pause the animation.
1	Resume the animation at its default speed.
2	Speed up the animation.
3	Slow down the animation.
4	Choose the previous preset configuration from an arbitrary list. The name of the configuration is shown at the top of the screen.
5	Choose the next preset configuration from the list.
*	Generate a random configuration and animate until it stabilizes or dies. If it dies, generate a new random configuration.

The source code for this example is particularly well documented. See *toolkit*`\apps\Demo3D\src\com\superscape\m3g\wtksamples\life3d\Life3D.java`.

A.8.2 PogoRoo

`PogoRoo` shows you a kangaroo bouncing up and down on a pogo stick. To steer the kangaroo, use the arrow keys. Push up to go forward, down to go backward, and left and right to change direction. You might need to hold down the key to see an effect.

FIGURE A-14 Bouncing Kangaroo

A.8.3 `retainedmode`

The `retainedmode` MIDlet plays a scene file that shows a tireless skateboarder in an endless loop.

FIGURE A-15 Tireless Skateboarder

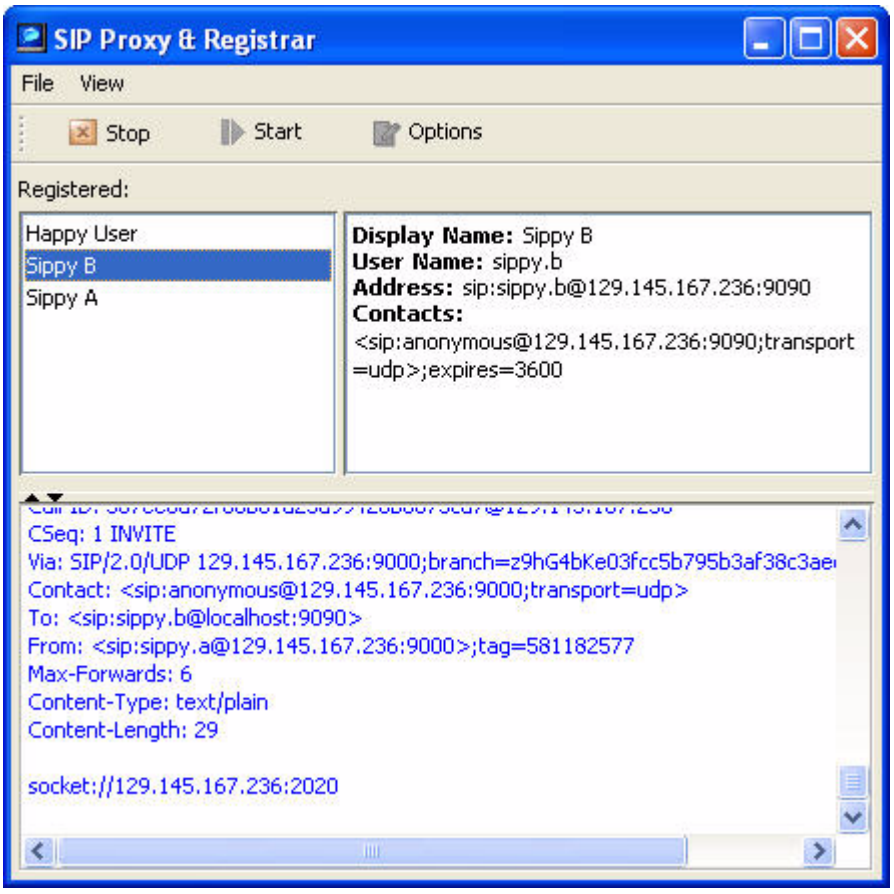
A.9 GoSIP

GoSIP is a chat application that uses SIP (JSR 180) to set up communications using a SIP proxy server and registrar.

Begin by running the SIP server. Choose File > Utilities. Select Start SIP Server and press Launch. The SIP proxy server window appears. Click Start to run the server.

Next, run two instances of the emulator with the `GoSIP` application.

FIGURE A-16 SIP Proxy and Registrar



In the first emulator, launch Sippy A. Enter your local machine name or IP address when you are prompted for the proxy host and choose Next, then Register. In the SIP server window, SIP messages from the emulator appear. `Sippy A` appears in the list of registered users. The emulator suggests you invite your friend Sippy B to talk. Don't do it yet.

In the second emulator, launch Sippy B. Just as before, enter the address of the SIP proxy, choose Next, then Register. The Sippy B user appears in the SIP server window.

In the first emulator, choose Invite. The second emulator indicates that it's ringing. Choose Answer to start the chat. Both emulators now show a Talking screen. You can send messages back and forth using the Send command.

When you are finished, choose Bye to end the chat.

A.10 i18nDemo

This MIDlet suite shows off the JSR 238 Mobile Internationalization API. The MIDlets String Comparator and Formatter show how to sort strings and display numbers appropriately for different locales. The third MIDlet, MicroLexicon, is a small phrase translator that comes in handy if you need to ask for a beer in Prague, Herzliya, Beijing, Milan, or several other locations.

Note – The default fonts for the Sun Java™ Wireless Toolkit for CLDC do not support Chinese and Japanese. To use these languages, follow these steps before running this demo:

1. Install a True Type font that supports Chinese or Japanese.
2. Modify *toolkit*`\wtklib\devices\`*skin-directory*`\`*skin*`.properties` to specify that font.

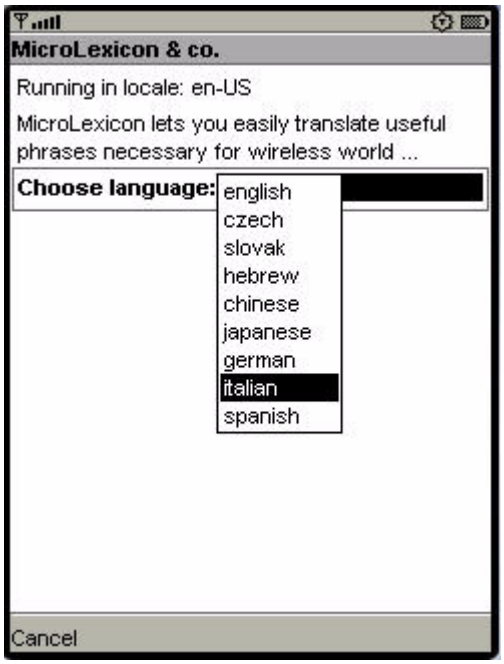
To run a MIDlet, use SELECT to highlight the MIDlet, then use the lower right button to Launch the MIDlet.

The String Comparator MIDlet demonstrates how strings (city names) are sorted differently depending on locale. Launch the MIDlet. Use the lower right button to view the menu. Click or Type 2 to select Sort - default, and the list is sorted alphabetically. Click or Type 3 to select Sort - slovak. It's easy to see the difference in the cities that begin with the letter Z, with and without the mark on top. Click Exit to return to the list of MIDlets.

The second MIDlet, Formatter, simply displays times and numbers formatted for different locales. Click next to view all four screens. Click Exit to return to the list of MIDlets.

The final MIDlet, MicroLexicon, translates phrases from one language to another language. To select the target language from the list, use the navigation arrows to highlight Choose Language. Click SELECT to view the language drop down. Use the navigation arrows to choose a language (see FIGURE A-17) and then click SELECT.

FIGURE A-17 Choosing the Target Language

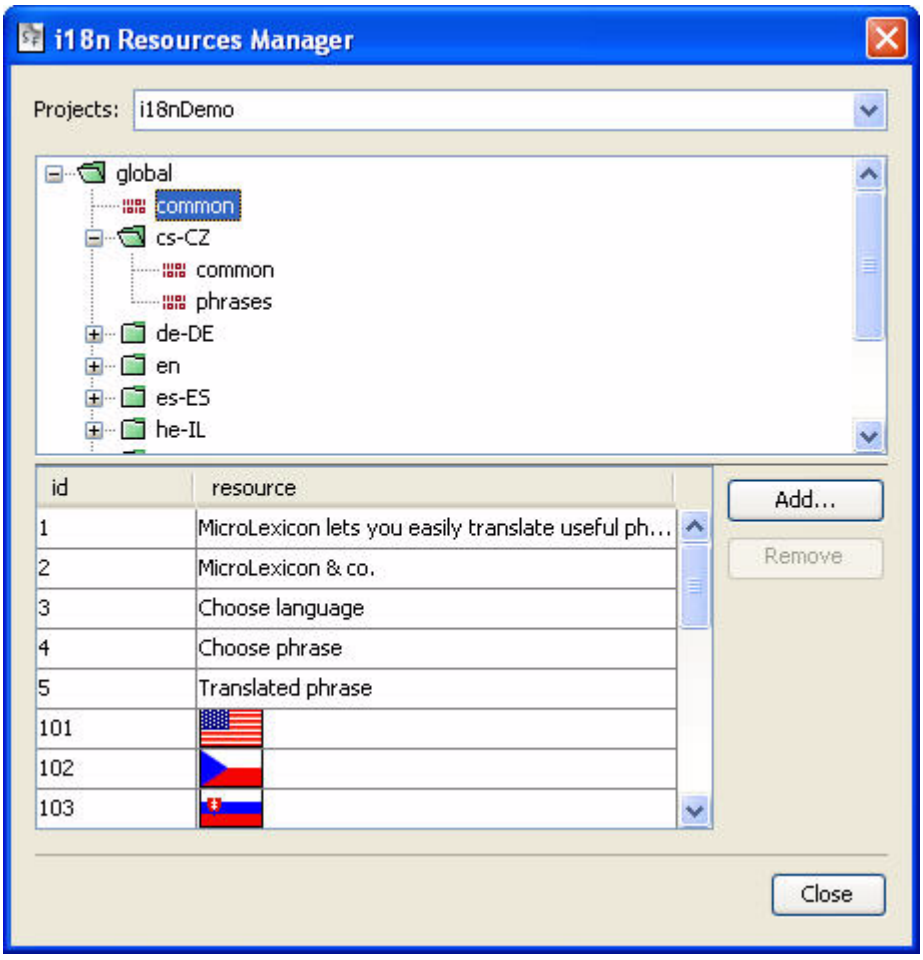


MicroLexicon displays a list of phrases. Highlight one and press the SELECT button on the emulator. You see the flag of the target language and the translated phrase.

To change the source language, choose Edit > Preferences. Click the i18n tab and enter a valid locale string. The next time you run the emulator and MicroLexicon, the instruction text appears in the given locale, if it is supported. One example that works is `cs-CZ`.

MicroLexicon is powered by MIDlet resources. To understand how you can use the toolkit to localize an application, choose Project > i18n Resources Manager. All the resources, both text and images, used by MicroLexicon, appear. You can edit the resources and run MicroLexicon again to see what happens. You don't need to build the application again because the resources are loaded at runtime.

FIGURE A-18 Internationalization Resources Manager



The resources themselves are stored in *workdir*`\apps\i18nDemo\res\global`.

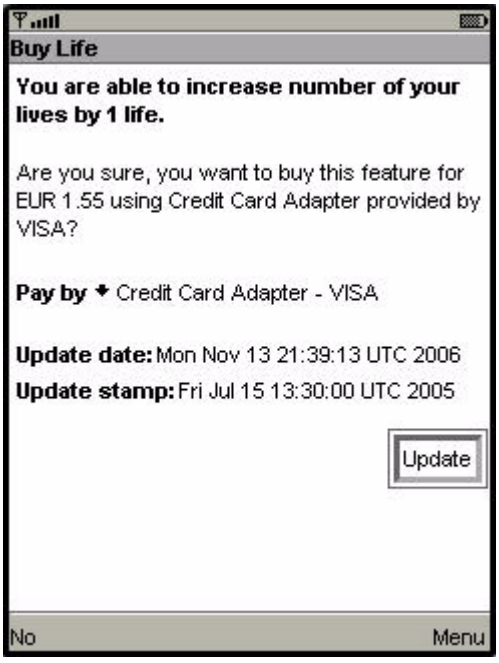
A.11 JBricks

JBricks is a game that demonstrates the use of the JSR 229 Payment API. The game itself resembles Breakout or Arkanoid. In JBricks, you can buy another life or a new game level. Behind the scenes, the Payment API handles the details.

To use the payment features of JBricks, use Project > Run via OTA to install JBricks into the emulator. If you don't know how to do this, read about it in Section 2.3.2, "Install" on page 2-8.

To see how `JBricks` uses the Payment API, choose either Buy Life or Buy Level from the game's main menu. Next, choose whether you want to buy a single life or three lives for a reduced price. The next screen gives you a choice of payment types.

FIGURE A-19 Choosing a Payment Type



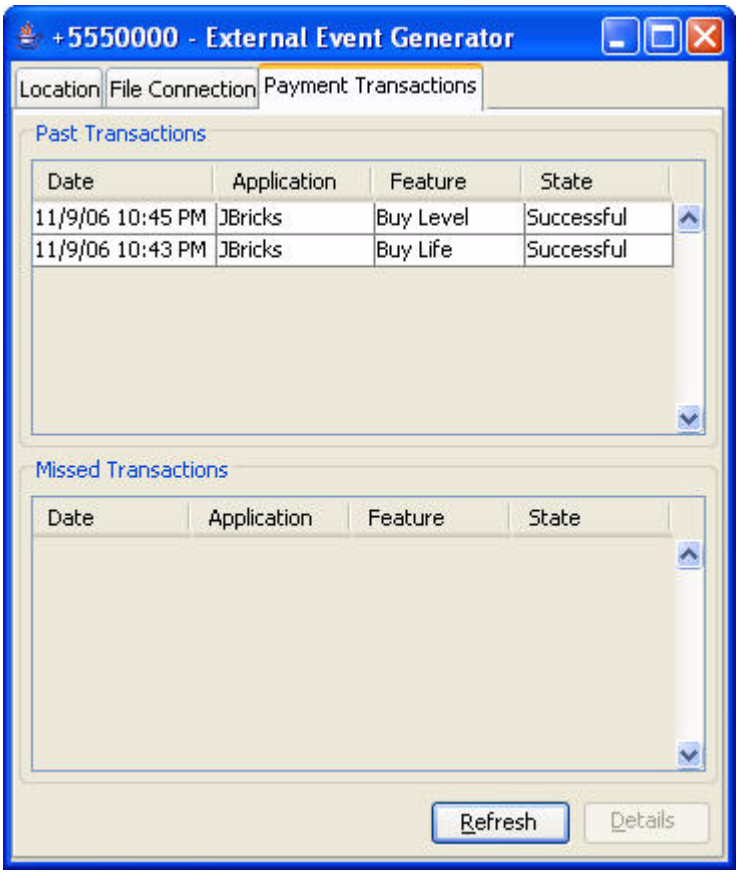
Use the navigation arrows to select the line starting with Pay by. Click the SELECT button to see the possible credit card adaptors in a drop down menu. Use the navigation arrows to select the VISA adaptor, then click SELECT. Click Yes on the lower right to proceed.

Next, you will be able to enter credit card information. Use any valid VISA number (for example, 4111111111111111) and a valid expiration date.

FIGURE A-20 Providing Payment Information

To view the transactions for the current instance of the emulator, choose MIDlet > External Events and click on the Payment Transactions tab. Transactions for this specific instance of the emulator appear.

FIGURE A-21 Viewing Transactions



In addition, you can view all transactions passing through the toolkit's payment system. Choose File > Utilities, then select Payment Console. A transaction in the console looks something like the following:

```
PSP Console running, using phone number +5550001.
PSP Server running at https://localhost:-1
Received Payment Request from 127.0.0.1
   Credit card issued by: VISA
   Credit Card type: 0
   Credit Card Number: 4111111111111111
   Credit Card Holder: Jonathan Knudsen
   Feature ID: 3_lives
   Credit Card Verification Number (CCV): 123
   Payload: null
Response to 127.0.0.1
```

```
HTTP/1.1 200 OK
Content-Length: 0
Pay-Response: SUCCESSFUL
Pay-Timestamp: 1156282954734
```

A.12 JSR172Demo

JSR172Demo shows how to access a web service from a MIDlet. The web service is already running on an Internet server. If you are behind a firewall, you must configure the emulator's proxy server settings. Choose Edit > Preferences, then select Network Configuration. Fill in the proxy server address file and the port number. Build and run the example.

JSR172Demo contains a single MIDlet named Server Script. Launch it and follow the prompts. You can browse through simulated news headlines, all of which are retrieved from the web service.

To see what is going on behind the scenes, use the network monitor.

A.13 MobileMediaAPI

The MobileMediaAPI application contains four MIDlets that showcase the toolkit's multimedia capabilities. This section describes the MIDlets and includes additional information about using multimedia from your applications.

A.13.1 Simple Tones

The Simple Tones example demonstrates how to use interactive synthetic tones. Select an example, then click Play on the lower right.

- Short Single Tone and Long Single Tone use `Manager.playTone()` to play tones with different pitch.
- Short MIDI event plays a chord on the interactive MIDI device (locator `"device://midi"`). The `shortMidiEvent()` method of `MIDIControl` is used to trigger the notes of the chord.
- To run the MMAPI Drummer demo, click or type number keys (0-9). Each number plays a different sound.

A.13.2 Simple Player

The Simple Player application demonstrates the range of audio and video capabilities of the emulator. It includes sample files in a variety of formats and can play files from the emulator's persistent storage or from HTTP URLs.

The player portion uses a generic `javax.microedition.media.Player` interface. The player displays duration, media time, and controls for running the media file. If metadata is available in a file, the player enables you to view the information, such as author and title. In the case of MIDI files, if karaoke text is present in the file, it displays on the screen during play. Graphical user interface controls can be viewed on the display screen if applicable. You can access these controls by selecting one of the media samples in Simple Player, then pressing the Menu button to view and select the desired command.

Select Simple Player then click Launch. The demo includes the following media samples:

- Bong plays a short WAV file. You can adjust certain playback features, as described later in this document. The display shows the duration of the sound in minutes:seconds:tenths of a second, for example 00:17:5. This audio sample is a resource file in the MIDlet suite JAR file.
- MIDI Scale plays a sample musical scale. The display shows the title of the selected music file, the duration of the song, the elapsed time during playback, and the current tempo in beats per minute (bpm). This MIDI file is stored in the MIDlet suite JAR file.
- Simple Ring Tone plays a short sequence of Beethoven's Fifth Symphony. The display shows the title of the selected music file, the duration of the song, the elapsed time in seconds and tenths of a second during playback, and the current tempo in beats per minute (bpm). This ringtone file (.jts format) is stored in the MIDlet suite JAR file.
- WAV Music plays a brief audio file. The display shows the title of the audio file, the duration of the audio the elapsed time during playback, and the playback rate in percent. This WAV file is retrieved from an HTTP server.
- MIDI Scale plays a MIDI file that is retrieved from an HTTP server.
- The Animated GIF example shows an animated GIF that counts from 1 to 5. The file is stored in the MIDlet suite JAR file.
- Audio Capture from a default device lets you capture audio from a microphone or connected device. The sound is captured and played back on the speaker. To avoid feedback, use a headset.
- Video Capture Simulation simulates viewing input video such as might be possible on a device equipped with a camera.
- MPEG1 Video [http]. Plays an MPEG video found at `http://java.sun.com/products/java-media/mma/media/test-mpeg.mpg`.

- `[enter URL]` allows you to play back media files from arbitrary HTTP servers. Type a valid URL (for example, `http://java.sun.com/products/java-media/mma/media/test-wav.mpg`) at the insertion point and click OK to play a file. If you want to open an HTTP directory from which to select media, be sure to add a slash to the end of the URL.

In addition, Simple Player parses ring tones in Ringing Tones text transfer language (RTTTL). See `http://www.convertyourtone.com/rtttl.html` for information on RTTTL.

The Simple Player includes a common set of commands that control media playback. The commands are available from the Simple Player menu, and some have associated keypad buttons. The following table describes these commands.

TABLE A-3 Simple Player Commands

Command	Key	Description
Mute/Unmute	0	Turns off sound but the file continues to play. This command toggles to Unmute.
Volume	* and #	Increases or decreases loudness.
META Data		Displays information provided by the media file such as copyright information, title, and track list.
Stop in 5 seconds		Pauses the audio play in five seconds when set during playback.
Rate	4 and 6	Alters the rate of speed of playback.
Tempo		Increases or decreases the tempo of the tone sequence or MIDI file.
Pitch	up and down	Lowers or raises the notes in a MIDI file.
Start/Stop Recording		Records the audio playback. A file is created containing the recorded audio in the directory in which the emulator is running. If you do not specify a filename, a file called recording.wav is created. This command toggles to Stop Recording.
Step Frame	7 and 9	Jumps forward or backward one frame at a time in a video file.
Play/Stop	2 and Select	Starts or stops the media.
Loop Mode		Plays back the audio file immediately after completion of play. Running Loopmode once plays the audio file once. Pressing a second time plays the file three times. Pressing a third time plays the file repeatedly. Pressing a fourth time returns to single play.

TABLE A-3 Simple Player Commands (Continued)

Command	Key	Description
Skip	1 and 3	Skips forward or backward five percent of the duration of the media file. The sound track syncs to the video.
Rewind		Returns to the start time of the audio playback.
Stop and Rewind	5	Stops playback and rewinds to the start time.
Quick Help		Displays a list of commands and keypad buttons.

The commands may or may not be available depending on the media type that Simple Player is playing. In addition, some commands can be invoked using the keypad buttons. The following table describes the availability of commands, their keypad equivalents, and the relevant class from MMAPI.

Note that a short list of commands and the corresponding keypad buttons is available in the Simple Player application itself. Just choose the Quick Help command from the menu.

A.13.3 Video

The Video application illustrates how the emulator is capable of playing animated GIF files and capturing video. On a real device with a camera, video capture can be used to show the user what the camera sees.

Animated GIFs and video capture can be implemented using either a `Form Item` or a `Canvas`. The Video demonstration includes all the possibilities. Animated GIF - Form [jar] shows an animated GIF as a Form Item. The form also includes some information about the playback, including the current time. Choose the Snapshot command to take a snapshot of the running animation. The snapshot will be placed in the form following the animated GIF.

- Animated GIF - Canvas [jar] shows an animated GIF in a Canvas. A simple indicator shows the progress through the animation. Choose Snapshot to get a still image of the current appearance. The snapshot is shown briefly, then the display goes back to the animation.
- Video Capture - Form simulates capturing video from a camera or other source and showing it as an Item in a Form. Choose the Snapshot command to take a snapshot of the captured video. The snapshot will be placed in the form following the video capture.
- Video Capture - Canvas simulates capturing video from a camera or other source and showing it in a Canvas. Choose Snapshot to get a still image of the current appearance. The snapshot is shown briefly, then the display goes back to the video capture.

- MPEG1 Video - Form, MPEG1 Video - Canvas

  The MPEG1 applications obtain MPEGs from the web, so if you are behind a firewall, you must configure the emulator's proxy server settings.

  Choose Edit > Preferences, then select Network Configuration. Check Use proxy server. Fill in the proxy server address field and the port number. For this demo, select HTTP/1.0.

  When you play the demo, expect to wait a few seconds while WTK obtains the data. The MPEG1 demos have the same behavior as Video Capture - Form and Video Capture - Canvas, respectively.

A.13.4 Pausing Audio Test

This MIDlet exists to demonstrate how the Sun Java™ Wireless Toolkit for CLDC will warn you if a paused MIDlet has not stopped its running Players. After you launch the MIDlet, choose the Play command to start playing some audio. The screen displays a status, which is either "Well-behaved" or "Not Well-Behaved."

Choose MIDlet > Pause from the emulator window's menu. As expected, the MIDlet is paused and no message is displayed on the toolkit console. Choose MIDlet > Resume from the emulator window's menu.

Now choose the Misbehave command. Pause the MIDlet again. In the toolkit console, you see the warning: `An active media (subtype Player) resource was detected while the MIDlet is paused. Well-behaved MIDlets release their resources in pauseApp().`

A.13.5 Attributes for MobileMediaAPI

The `MobileMediaAPI` applications have the following attributes that you can modify in the project settings dialog box User Defined tab:

TABLE A-4 Descriptions of MMAPI-specific MIDlet attributes

Attribute	Description
`PlayerTitle-`*n*	Name of the *n*th media title to be played back by the Simple Player MIDlet.

TABLE A-4 Descriptions of MMAPI-specific MIDlet attributes (Continued)

Attribute	Description
`PlayerURL-`*n*	Location of the *n*th media title, `PlayerTitle-`*n*, to be played back by the Simple Player MIDlet.
`VideoTest-`*n*	The name of the *n*th media title to be played back by the `Video` application.
`VideoTest-URL`*n*	Location of the *n*th media title, `VideoTest-`*n*, to be played back by the `Video` application.

A.14 Network Demo

This demo has two MIDlets: Socket Demo and Datagram Demo. Each demo requires you to run two emulator instances so that you can emulate the server and client relationship.

A.14.1 Socket Demo

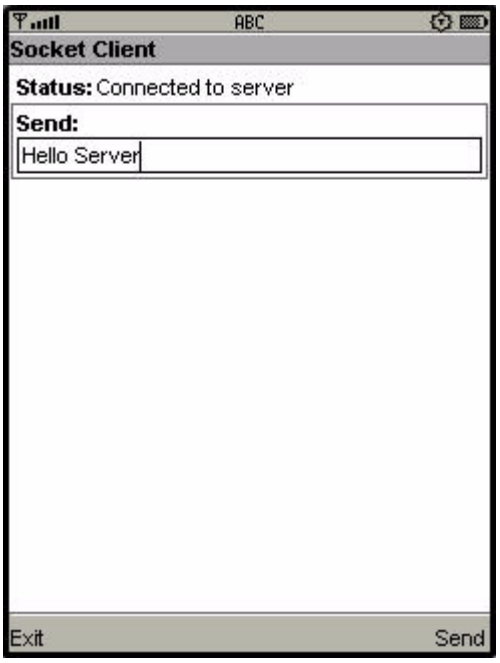
Run two instances of the emulator. One acts as the socket server, and the other as the socket client.

In the first emulator, launch the application, then select the Server peer. Choose Start. The emulator explains that the demo wants to send and receive data over the network and asks, “Is it OK to use network?” Choose Yes. The Socket Server displays a screen that indicates it is waiting for a connection.

In the second emulator, launch the application, select the Client peer, then choose Start. The emulator explains that the demo wants to send and receive data over the network and asks, “Is it OK to use network?” Choose Yes. The Socket Client displays a screen that indicates it is connected to the server. Use the down navigation arrow to highlight the Send box. Type a message in the Send box, then choose the Send soft key.

For example, in the client, type `Hello Server` In the Send box (see FIGURE A-22). Choose the Send soft key. The emulator activates a blue light during the transmission.

FIGURE A-22 Sending a Message from the Socket Client



On the emulator running the Socket Server, the Status reads: `Message received - Hello Server`. You can use the down arrow to move to the Send box and type a reply. For example, `Hello Client, I heard you`. Select Send. See FIGURE A-23.

FIGURE A-23 Server Shows Message Received and Message to Send

Back in the Socket Client, the status shows the message received from the server. Until you send a new message, the Send box contains the previous message you sent.

A.14.2 Datagram Demo

This demo is similar to Socket Demo.

Run two instances of the emulator. One acts as the datagram server, and the other as the datagram client.

In the first emulator, launch Datagram Demo, then select the Server peer. Choose Start. The emulator explains that the demo wants to send and receive data over the network and asks, "Is it OK to use network?" Choose Yes. Initially, the Datagram Server status is `Waiting for connection`, and the Send box is empty.

In the second emulator, launch Datagram Demo, select the Client peer, then choose Start. The emulator explains that the demo wants to send and receive data over the network and asks, "Is it OK to use network?" Choose Yes. The Datagram Client status is: `Connected to server`. Use the down navigation arrow to highlight the Send box. Type a message in the Send box, then choose the Send soft key. For example, type `Hello datagram server.`

On the emulator running the Datagram Server, the Status displays: `Message received - Hello datagram server`. You can use the down arrow to move to the Send box and type a reply to the client.

In the Datagram Client, the status field displays the message received from the server. The Send box contains the last message you sent.

A.15 ObexDemo

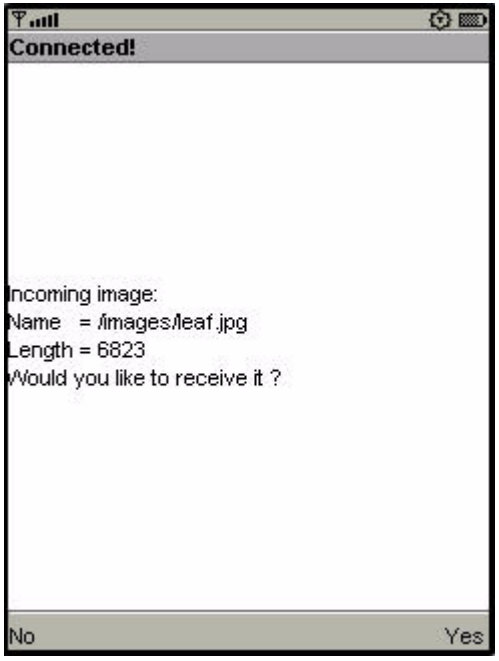
This application shows how to transfer image files between emulator instances using the OBEX API. This demonstration shows the use of OBEX over a simulated infrared connection.

Run two instances of the emulator. One listens for incoming connections, while the other attempts to send an image. In the first emulator, launch the application then choose Obex Demo, then Receive Image. The emulator explains that an OBEX connection allows other devices to talk to yours and asks, “Is it OK to make the connection?“ Choose Yes. The listener emulator displays a screen that indicates it is waiting for incoming connections.

In the second emulator (the sender), launch Obex Demo, then choose Send Image. You see a list of images. Select one and choose Send. The emulator explains the demo wants to make an outgoing client connection, and asks if it is OK. Choose Yes. The Send Image utility uploads the image.

In the listening emulator, the utility displays information about the incoming image and asks “Would you like to receive it?“ See FIGURE A-24.

FIGURE A-24 Listener Prompting to Accept a Connection



Choose Yes. The image you selected is transferred over the simulated infrared link and displayed on the first emulator. See FIGURE A-25.

FIGURE A-25 Successfully Transferred Image

A.16 PDAPDemo

`PDAPDemo` shows how to use the PIM and FileConnection APIs that are part of the JSR 75 specification.

A.16.1 Browsing Files

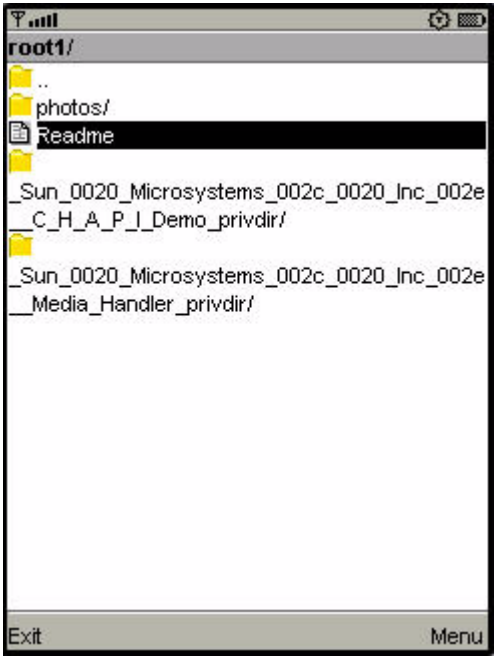
To run the file browser, you'll need to give the MIDlet appropriate security authorization, if you have not already done so. Choose Edit > Preferences. Click on the Security tab. Change the Security domain to maximum and press OK.

Now open and run the `PDAPDemo` project. Launch the `FileBrowser` MIDlet. You see a directory listing, and you can browse through the available directories and files. By default there is one directory, `root1`.

FIGURE A-26 Browsing Files

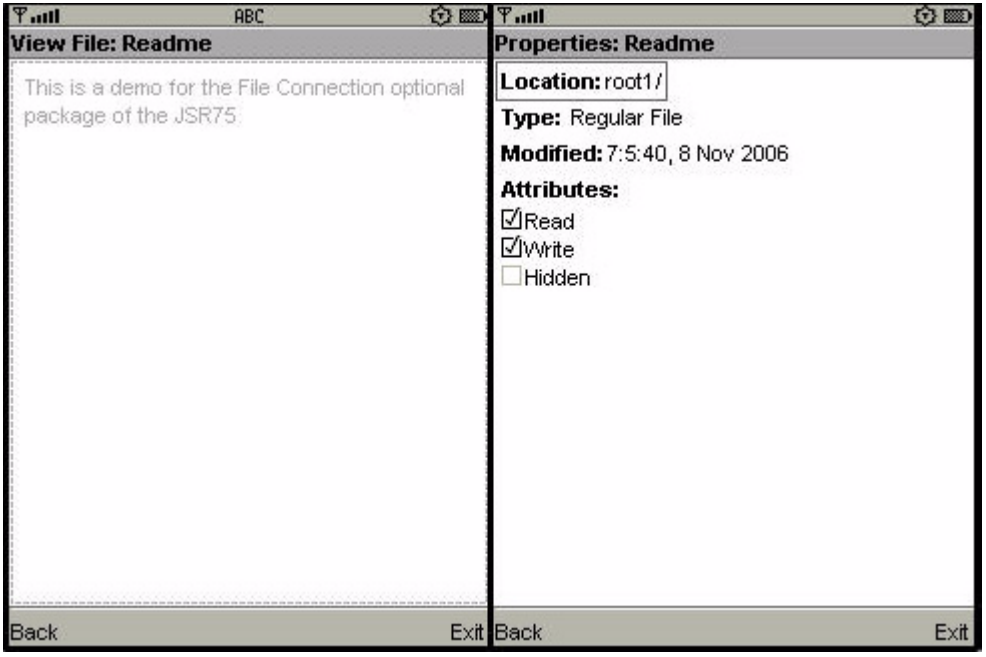
Select the directory and press the select button to enter it.

FIGURE A-27 Contents of the `root1` Directory



Using the commands in the demonstration, you can view the file or see its properties. Try selecting the file and choosing Properties or View from the menu.

FIGURE A-28 Viewing File Contents and File Properties



The actual files are located in *workdir*`\appdb\DefaultColorPhone\filesystem`, assuming you are using the `DefaultColorPhone` emulator skin. You can add files and root directories as you wish and they will be visible to the JSR 75 File API. See Chapter 10 for more information.

A.16.2 The PIM API

The `JSR75 PIM APIs` example demonstrates how to access personal information, like contact lists, calendars, and to-do lists. After you launch the example, choose a type of list from the main menu.

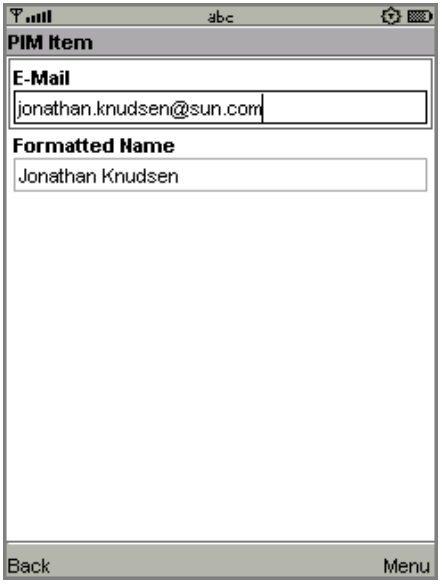
In this example application, each type of list works the same way and each list type contains a single list. For example, if you choose Contact Lists, there is a single contact list called Contacts. Event Lists contains a single list called Events, and To Do Lists contains a single list named To Do.

FIGURE A-29 Choosing a List Type

Once you've selected a list type and chosen the specific list, you can view all the items in the list. If this is the first time you've run the example, the list is probably empty.

To add an item, choose New from the menu. The application prompts you for a Formatted Name for the item. You can add more data fields to this item using Add Field in the menu. You see a list of field names. Pick one, then enter the value for the new field.

FIGURE A-30 Adding Contact Fields



To save the list item, choose Commit (option 3) from the menu.

You can return to the list by choosing the Back command. You'll see the item you just created in the list.

The items that you create are stored in standard vCard or vCalendar format in the *workdir*`\appdb\`*skin*`\pim` directory. See Chapter 10 for more information.

The PIM API allows for exporting contact, calender, and to-do items in a standard format. The exact format depends on the list type. When you are viewing an item in any list, the menu contains a command for viewing the exported item.

For example, when you are viewing a contact list item, the menu contains Show vCard. When you choose this command, the exported item is shown on the screen. Calendar items and to-do items both get exported as vCalendar.

A.17 SATSADemos

`SATSADemos` includes demonstrations of SATSA, the Security and Trust Services APIs. Most of the demonstrations show how to communicate with a smart card. The emulator can communicate with a simulated smart card using a socket protocol. The smart card simulator, `cref`, is included with the toolkit. See Chapter 14 for details.

The following sections contain instructions for each menu choice for this demo. For each demo, be sure to do the following *before* launching the emulator:

- Run the instance(s) of `cref` from the command line.
- Be sure to set the set the security domain to maximum.

A.17.1 APDUMIDlet

This MIDlet demonstrates communication with a smart card using Application Protocol Data Units (APDUs), small packets of data. `APDUMIDlet` expects to find two simulated smart cards. You can run the smart card simulator using `cref`, which is part of the Java Card Development Kit.

The `Mohair` application includes pre-built memory images that you can use with `cref`. The memory images contain Java Card applications with which `Mohair` interacts. The memory images are in the root directory of the `Mohair` project.

On Windows, start up two instances of `cref` like this, one for each simulated card slot (assuming the current directory is the toolkit installation directory):

```
start bin\cref -p 9025 -i apps\SATSADemos\demo2.eeprom
start bin\cref -p 9026 -i apps\SATSADemos\demo2.eeprom
```

On Linux you can use:

toolkit`/bin/cref -p 9025 -i apps/SATSADemos/demo2.eeprom`
toolkit`/bin/cref -p 9026 -i apps/SATSADemos/demo2.eeprom`

Note that the port numbers (9025 and 9026 in this example) must match the port numbers you specified in the SATSA preferences, described in Chapter 14. Also, make sure you use the correct path to `demo2.eeprom`.

Once you have the two smart card simulators running, you can run `APDUMIDlet`.

A.17.2 SATMIDlet

`SATMIDlet` demonstrates smart card communication with a slight variation on APDU communication.

To set up the simulated smart card, use `cref`, very much like you did for `APDUMIDlet`. This time you don't have to specify a port number, and the memory image is different:

Windows:	`start bin\cref -i apps\SATSADemos\sat.eeprom`
Linux:	*toolkit*`/bin/cref -i apps/SATSADemos/sat.eeprom`

When the smart card simulator is running, you can run `SATMIDlet` to communicate with card applications.

A.17.3 CryptoMIDlet

`CryptoMIDlet` demonstrates the general cryptographic features of SATSA. It does not interact with a smart card in any way.

A.17.4 MohairMIDlet

`MohairMIDlet` has two functions. The first, Find slots, displays all the available card slots. Each slot has a number followed by 'C' or 'H' indicating whether the slot is cold-swappable or hot-swappable. After viewing the slots select Back to return to the first screen.

The second part of `MohairMIDlet`, SATSA-PKI Sign test, uses a smart card to generate a digital signature. As with the earlier demonstrations, you need to run `cref` with the right memory image to prepare for the connection from `MohairMIDlet`. Type the following in the installation directory:

Windows:	`start bin\cref -i apps\SATSADemos\sat.eeprom`
Linux:	*workdir*`/bin/cref -i apps/SATSADemos/sat.eeprom`

In the emulator, highlight SATSA-PKI Sign test and choose SELECT. The following confirmation message appears:

```
This certificate will be used: MohairAuth
```

Select the OK soft key.

For PIN 1, type: `1234`

Select the OK soft key. The following confirmation message appears:

```
This string will be signed: JSR 177 Approved
```

Select the OK soft key. The following confirmation message appears:

```
This certificate will be used: MohairAuth
```

Select the OK soft key.

For non repudiation key 1 PIN, type: `2345`

A.18 SATSAJCRMIDemo

This application contains a single MIDlet, `JCRMIMIDlet`, which shows how to communicate with a card application using Java Card RMI, a card-friendly remote object protocol. As with some of the MIDlets in `SATSADemos`, you need to start up `cref` with an appropriate memory image:

Windows:	`start bin\cref -p 9025 -i apps\SATSADemos\demo2.eeprom`
Linux:	*workdir*`/bin/cref -i apps/SATSADemos/demo2.eeprom`

Now run `JCRMIMIDlet` to see how your application can communicate with a distributed object on the card.

A.19 SIPDemo

This application is a very simple example of using SIP (JSR 180) to communicate directly between two devices. Usually devices will use SIP with a proxy server to set up direct communications of some kind. For a more complete example involving a proxy, take a look at `GoSip`.

To see how SIPDemo works, run two instances of the emulator. In the first, choose Receive message. You can use the default port, 5070, and choose Receive. The first emulator is now listening for incoming messages.

In the second emulator, choose Send message. Fill in values for the recipient, port number, subject, and message, or accept the defaults, and choose Send. Your message will be displayed in the first emulator. The first emulator's response is displayed in the second emulator.

Try it again with the network monitor turned on. You can see the communication between the emulators in the network monitor SIP tab.

A.20 SVGContactList

This application uses different skins to display the same contact list information and a news banner. The skins have different colors and fonts.

Select SVGContactlist(skin 1) or SVGContactlist(skin 2), then click Launch.

Use the up and down arrows to navigate the list of contacts. The highlighted name is marked with a special character (a > or a dot) and is displayed in a larger font.

FIGURE A-31 Contact List Shown with Skin 2

Press the select button to see more information for the highlighted name.

FIGURE A-32 Contact List Details

Press select again to return to the contact list.

A.21 SVGDemo

This suite contains MIDlets that demonstrate different ways of using the JSR 226 Scalable 2D Vector Graphics API for J2ME. This API provides ways to load manipulate, render, and play SVG content.

The Scalable Vector Graphics (SVG) 1.1 specification defines a language for describing two-dimensional graphics in XML. The full specification is available at `http://www.w3.org/TR/SVG11/`.

SVG Tiny (SVGT) is a subset of SVG that is appropriate for small devices like mobile phones. See `http://www.w3.org/TR/SVGMobile/`. SVG Tiny is a compact yet powerful XML format for describing rich, interactive, and animated 2D content. Graphical elements can be logically grouped and identified by the SVG markup.

A.21.1 SVG Browser

The SVGBrowser MIDlet displays SVG files residing in the phone file system. Before running this demo, place an SVG file in the directory *workdir*`\appdb\DefaultColorPhone\filesystem\root1`.

Launch the demo. The application displays the contents of `root1`. Select your SVG file and choose the Open soft key.

A.21.2 Render SVG Image

Render SVG Image loads an SVG image from a file and renders it. Looking at the demo code you can see that the image is sized on the fly to exactly fit the display area. The output is clear and sharp.

A.21.3 Play SVG Animation

This application plays an SVG animation depicting a Halloween greeting card. Press 8 to play, 5 to start, and 0 to stop. If you press 8, pressing 5 resumes the animation. If you press 0, pressing 5 starts the animation from the beginning.

The SVG file contains a description of how the various image elements evolve over time to provide this short animation.

In the following code sample, the JSR 226 `javax.microedition.m2g.SVGImage` class is used to load the SVG resource. Then, the `javax.microedition.m2g.SVGAnimator` class can take all the complexity of SVG animations and provides a `java.awt.Component` or `javax.swing.JComponent` which plays the animation. The `SVGAnimator` class provides methods to play, pause and stop the animation.

```
import javax.microedition.m2g.ScalableGraphics;
import javax.microedition.m2g.SVGImage;

...
String svgURI = ...;
SVGImage svgImage = (SVGImage) SVGImage.createImage(svgURI, null);
SVGAnimator svgAnimator = SVGAnimator.createAnimator(svgImage);

// If running a JSE applet, the target component is a JComponent.
JComponent svgAnimationComponent = (JComponent)
svgAnimator.getTargetComponent();
...

svgAnimator.play();
...
svgAnimator.pause();
...
svgAnimator.stop();
```

A.21.4 Create SVG Image from Scratch

This demo builds an image using API calls. It creates an empty SVGImage, populates it with a graphical content, and then displays that content.

A.21.5 Bouncing Balls

Bouncing Balls plays an SVG animation. Press 8 to play, 5 to start, and 0 to stop. If you press 8, pressing 5 resumes the animation. If you press 0, pressing 5 starts the animation from the beginning.

A.21.6 Optimized Menu

In this demo, selected icons have a yellow border. As you move to a new icon, it becomes selected and the previous icon flips to the unselected state. If you navigate off the icon grid, selection loops around. That is, if the last icon in a row is selected, moving right selects the first icon in the same row.

This demo illustrates the flexibility that combining UI markup and Java offers: a rich set of functionality (graphics, animations, high-end 2D rendering) and flexibility in graphic manipulation, pre-rendering or playing.

In this example, a graphic artist delivered an SVG animation defining the transition state for the menu icons, from the unselected state to the selected state. The program renders each icon's animation sequence separately into off-screen buffers (for faster rendering later on), using the JSR 226 API.

With buffering, the MIDlet is able to adapt to the device display resolution (because the graphics are defined in SVG format) and still retain the speed of bitmap rendering. In addition, the MIDlet is still leveraging the SVG animation capabilities.

The task of defining the look of the menu items and their animation effect (the job of the graphic artist and designer) is cleanly separated from the task of displaying the menu and starting actions based on menu selection (the job of the developer). The two can vary independently as long as both the artist and the developer observe the SVG document structure conventions.

A.21.7 Picture Decorator

In this demo you use the phone keys to add decorations to a photograph. The key values are:

1	key shrink
2	key next picture
3	key grow
4	key help
5	key horizontal flip
6	ley vertical flip
7	key rotate counter-clockwise
8	key previous picture
9	key rotate clockwise
#	display picker options

This demo provides 16 pictures for you to decorate.

Use the 2 and 6 keys to page forward and back through the photos.

To decorate, press # to display the picker. Use the arrow keys to highlight a graphic object. The highlighted object is enlarged. Press SELECT to choose the current graphic or press the arrow keys to highlight a different graphic. Press SELECT again to add the graphic to the photo. When the decoration is added you see a red + on the graphic. This means it is selected and can be moved, resized, and manipulated.

FIGURE A-33 Decorated Picture with Quotation Selected

Use the navigation arrows to move the graphic. Use 1 to shrink the graphic, and 3 to enlarge the graphic. Use 5 or 6 to flip, and 7 or 9 to rotate. When you are satisfied with the position, press SELECT. Note that a green triangle appears. This is a cursor. Use the navigation keys to move the green triangle around the picture. When the cursor is over an object it is highlighted with a red box. Press SELECT. The red + indicates the object is selected.

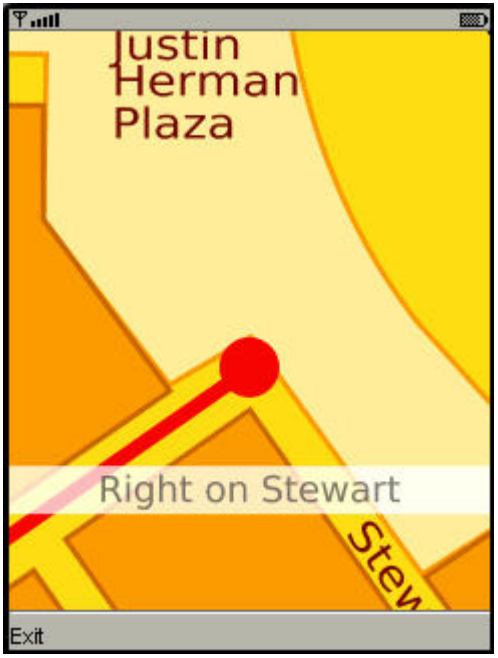
FIGURE A-34 Highlighted Mustache

To remove a decoration (a property), select an object, then click the Menu soft key. Press 2 to remove a property.

A.21.8 Location Based Service

Launch the application. A splash screen (also used as the help) appears. The initial view is a map of your itinerary - a walk through San Francisco. The bay (in blue) is on the right of your screen. Press 1 to start following the itinerary. The application zooms in on your location on the map. Turn-by-turn directions appear in white boxes on the horizontal axis. While the itinerary is running, Press 7 to rotate the map counter-clockwise. Note, the map rotates and the text now appears on the vertical axis. Press 7 again to restore the default orientation. Press 4 to display the help screen.

FIGURE A-35



A.22 WMADemo

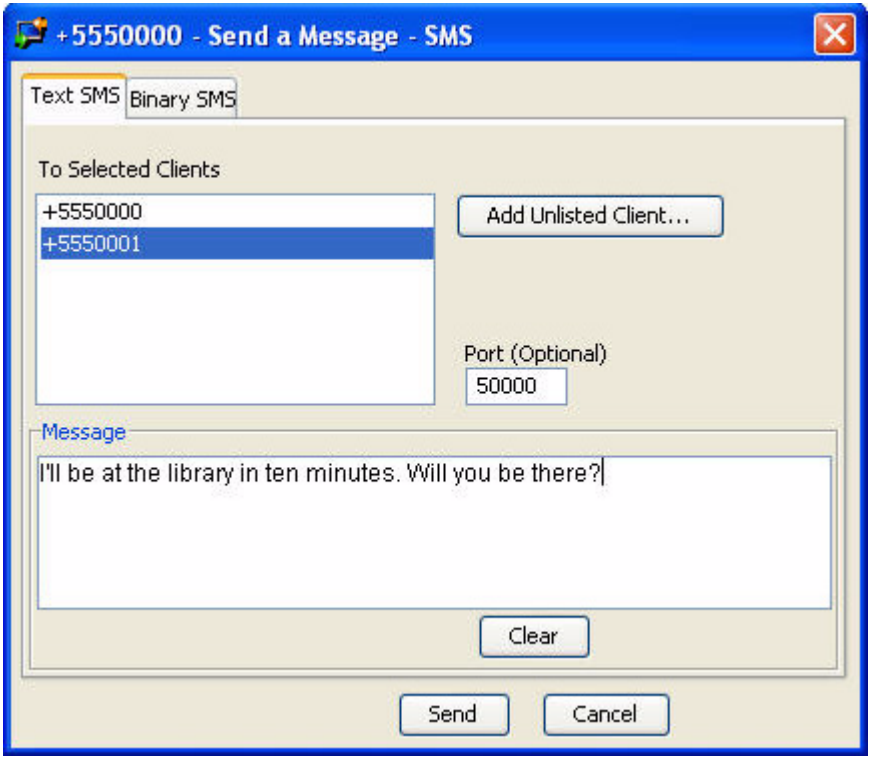
This application shows how to send and receive SMS, CBS, and MMS messages. The Sun Java™ Wireless Toolkit for CLDC offers a flexible emulation environment to support messaging. Messages can be exchanged between emulator instances and can be generated or received using the WMA console utility.

Because this example makes use of the push registry, you can't see all of its features just by using the Run button. Use the Run via OTA feature to install the application into the emulator in a process that mirrors how applications are installed on real devices. If you don't know how to do this, read about it in Chapter 2.

To exercise the push registry, use the WMA console to send the emulator a message. Launch the console by choosing File > Utilities. Click on the Open Console button in the WMA box to launch the WMA console.

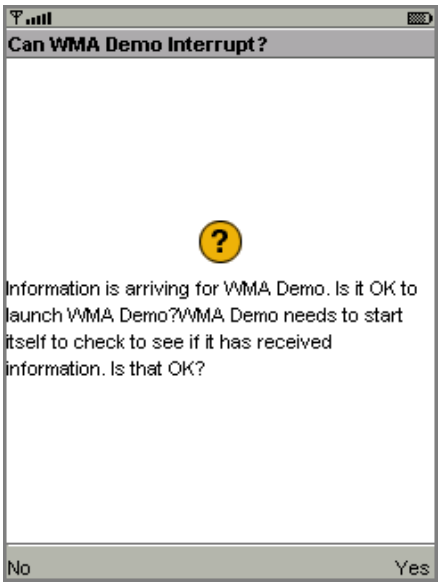
Click on the Send SMS... button in the WMA console window. Choose the number that corresponds to the emulator, probably +5550000. If you're not sure what number the emulator is using, look in its title bar. Choose the number in the SMS message window, then fill in a port number of 50000. Type your text message in the Message field and click on Send.

FIGURE A-36 Sending a Text Message



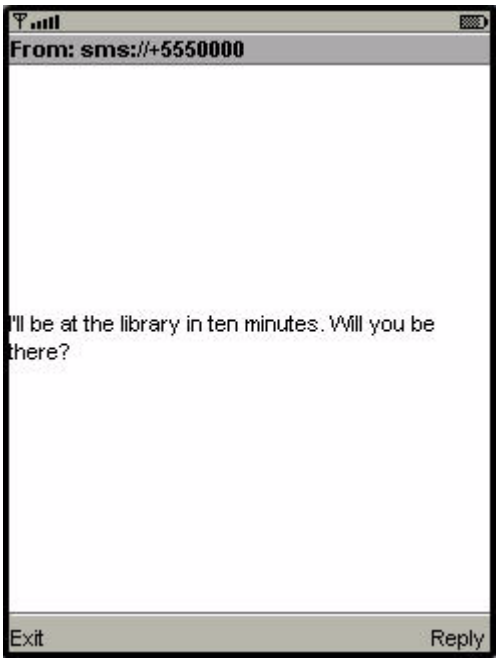
The emulator asks if it can launch the `WMADemo` application.

FIGURE A-37 Push Registry Message



Choose Yes. The `SMSReceive` MIDlet is launched and immediately displays the incoming SMS message.

FIGURE A-38 Incoming Text Message



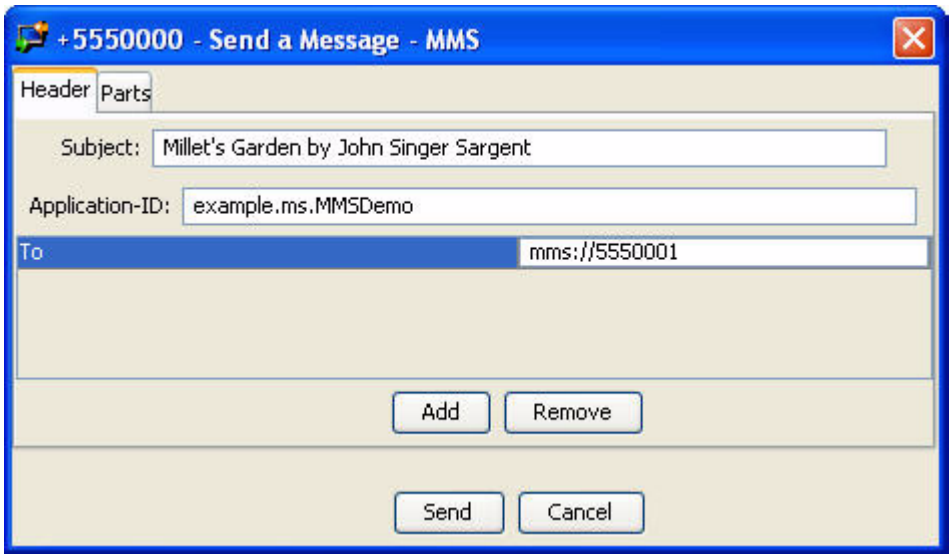
You can also use the WMA console to send and receive CBS and MMS messages. See Chapter 7 for more information.

Note – If you are attempting to send text messages to `WMADemo` using the WMA console, make sure to specify the port number as 50000. Use port 50001 for CBS messages. For MMS messages, use `example.mms.MMSDemo` as the application ID.

For example, to send an MMS message from the WMA console to the emulator, make sure that `WMADemo` has been installed using Run via OTA as described above. Launch the demo and choose MMS Receive.

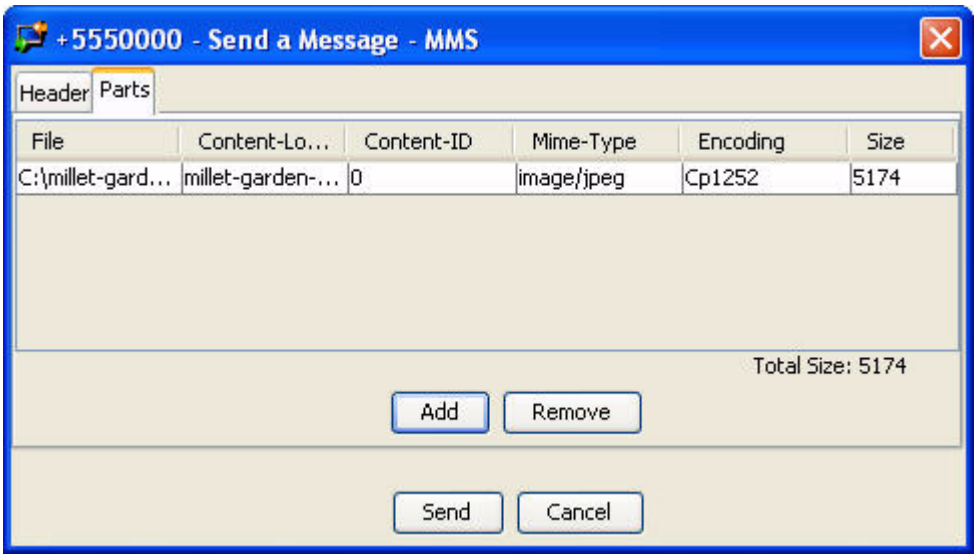
In the WMA console, click on Send MMS... to open the MMS composition window. Fill in a message subject, the application ID `example.mms.MMSDemo`, and the telephone number of the running emulator.

FIGURE A-39 Addressing an MMS message



Next, click on the Parts tab. The WMA console allows you to select files from your hard disk that you wish to send as parts of the MMS message. Click Add to add a file to the message. Use the file browser to find the file you want to send and click OK.

FIGURE A-40 Adding Parts to an MMS Message



Click on Send to send the message.

The emulator asks if it can launch WMADemo. Click on Yes. The image and its information are displayed.

FIGURE A-41 `WMADemo` Receives the Image

APPENDIX **B**

Command Line Reference

This appendix describes how to operate the Sun Java™ Wireless Toolkit for CLDC from the command line and details the steps required to build and run an application. It also describes the Sun Java™ Wireless Toolkit for CLDC certificate manager utility, called `MEKeyTool`, and the MIDlet signing utility, called `JadTool` (Java Application Descriptor Tool).

B.1 Prerequisites

Before building and running an application from the command line, verify that you have a version no earlier than 1.4.2 of the Java SE software development kit. Make sure the `jar` command is in your path. To find the version of the development kit, run the `jar` command and then run `java -version` at the command line.

For more examples, see the files `build.bat` and `run.bat` in the `bin` directories of the demonstration applications. You can find these files in:

Windows:	*toolkit*`\apps\`*demo*`\bin`
Linux:	*toolkit*`/apps/`*demo*`/bin`

toolkit is the installation directory of the Sun Java™ Wireless Toolkit for CLDC and *demo* is the name of one of the demo applications.

B.2 The Development Cycle

For a full description of developing MIDP applications, see Chapter 2. This section describes how to accomplish each of the steps in the development cycle from the command line.

B.2.1 Build

In the user interface, building a project is a single step. Behind the scenes, however, there are actually two steps. First, Java source files are compiled into Java class files. Next, the class files are *preverified*, which means they are prepared for the CLDC KVM.

Use the `javac` compiler from the Java SE development kit to compile Java source files. You can use the existing Sun Java™ Wireless Toolkit for CLDC project directory structure. You'll need to use the `-bootclasspath` option to tell the compiler to use the MIDP APIs, and you'll use the `-d` option to tell the compiler where to put the compiled class files.

The following example shows how you could compile a MIDP 2.0 application, taking source files from the `src` directory and placing the class files in the `tmpclasses` directory. Newlines have been added for clarity.

Windows

```
javac
  -bootclasspath ..\..\lib\cldcapi10.jar;..\..\lib\midpapi20.jar
  -d tmpclasses
  src\*.java
```

Linux

```
javac
  -bootclasspath ../../lib/cldcapi10.jar;../../lib/midpapi20.jar
  -d tmpclasses
  src/*.java
```

If you want to use the optional APIs that are supported by the toolkit, add their JAR files to the `-bootclasspath` option.

For more information on `javac`, consult the Java SE documentation.

The next step is to preverify the class files. In the `bin` directory of the Sun Java™ Wireless Toolkit for CLDC lives a handy utility called `preverify`. The syntax for the preverify command is as follows:

`preverify [`*options*`] `*files* | *directories*

Some of the options are as follows:

`-classpath` *classpath*

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

`-d` *output-directory*

Specify the target directory for the output classes. This directory must exist before preverifying. If this option is not used, the preverifier places the classes in a directory called `output`.

Following the example for compiling, use the following command to verify the compiled class files. As before, newlines are added for clarity.

Windows

```
preverify
  -classpath ..\..\lib\cldcapi10.jar;..\..\lib\midpapi20.jar
  -d classes
  tmpclasses
```

Linux

```
preverify
  -classpath ../../lib/cldcapi10.jar;../../lib/midpapi20.jar
  -d classes
  tmpclasses
```

As a result of this command, preverified class files are placed in the `classes` directory. If your application uses WMA, MMAPI, or other versions of CLDC or MIDP, be sure to include the relevant `.jar` files in the classpath.

B.2.2 Package

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

Create a manifest file containing the appropriate attributes as specified in the MIDP specification. You can use any 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-2.0
```

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 Java SE software development kit. The syntax is as follows:

`jar cfm` *file manifest* `-C` *class-directory* `. -C` *resource-directory* `.`

The arguments are as follows:

- *file* - JAR file to create.
- *manifest* - Manifest file for the MIDlets.
- *class-directory* - Directory containing the application's classes.
- *resource-directory* – Directory containing the application's resources.

For 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 .
```

Create a JAD file containing the appropriate attributes as specified in the MIDP specification. You can use any 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.

For example, a JAD file might have the following contents:

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

B.2.3 Run

You can run the emulator from the command line. The Sun Java™ Wireless Toolkit for CLDC's `bin` directory contains the command `emulator`. The syntax for the `emulator` command is as follows:

`emulator` *options*

The general options are as follows:

- `-help` - Display a list of valid options.

- `-version` - Display version information about the emulator.
- `-Xquery` - Print emulator skin information on the standard output stream and exit immediately. The information includes the skin name, screen size, and other capabilities.

Options that pertain to running MIDlet suites are as follows:

- `-Xdevice:`*skin-name* - Run an application on the emulator using the given skin name. For a list of skin names, see Section 4.1, "Emulator Skins" on page 4-1.
- `-Xdescriptor:`*jad-file* - Run an application locally using the given JAD file.
- `-classpath` *classpath* - Specify the classpath for libraries required to run the application. Use this option when running an application locally.
- `-Dcom.sun.midp.io.http.proxy` - Set the HTTP and HTTPS proxy servers at run time. For example:

  `-Dcom.sun.midp.io.http.proxy=`*proxy-host*`:`*proxy-port*
- `-Dcom.sun.midp.midlet.platformRequestCommand` - Specify the browser to use when applications call a URL. For example:

  `-Dcom.sun.midp.midlet.platformRequestCommand=`*browser*

  A sample call might look like:

  `emulator -Dcom.sun.midp.midlet.platformRequestCommand=firefox -Xjam:install=`*URL-to-app-using-platformRequest-method*

  If you want to use the same browser every time, you can add the following line to *toolkit*`/lib/system.config`:

  `com.sun.midp.midlet.platformRequestCommand:` *browser*

  Windows: In Windows, if this parameter is not specified, the default browser is used.

  Linux: For Linux, this parameter is required because Linux systems do not usually have a default browser. If it is missing, nothing happens when an application tries to open a URL.
- `-Xjam:`*command=application* - Run an application remotely using the Application Management Software (AMS) to run using OTA provisioning. If no application is specified with the argument, the graphical AMS is run. The commands are as follows:

  `install=`*jad-file-url* | `force` | `list` | `storageNames`|

  Install the application with the specified JAD file onto a device.

  Also

  `run=[`*storage-name* | *storage-number*`]`

  Run a previously installed application. The application is specified by its valid storage name or storage number.

`remove=[`*storage-name* | *storage-number* | `all]`

Remove a previously installed application. The application is specified by its valid storage name or storage number. Specifying `all`, all previously installed applications are removed.

- `transient=`*jad-file-url* **-** Install, run, and remove the application with the specified JAD file. Specifying `transient` causes the application to be installed and run and then removed three times.

B.2.4 Debugging

You can use the following options with the emulator for debugging and tracing.

- `-Xverbose:`*trace-options* **-** Display trace output, as specified by a list of comma-separated options, as follows:
  - `gc` - Trace garbage collection
  - `class` - Trace class loading
  - `all` **-** Use all tracing options
- `-Xdebug` **-** Enable runtime debugging. The `-Xrunjdwp` option must also be used.
- `-Xrunjdwp:`*debug-settings* **-** Start a Java debug wire protocol session, as specified by a list of comma-separated debug settings. The `-Xdebug` option must also be used. Valid debug settings include the following:
  - `transport=`*transport-mechanism* **-** Transport mechanism used to communicate with the debugger. The only transport mechanism supported is `dt_socket`.
  - `address=`*host:port* **-** 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}* **-** Starts 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).

B.3 Launching Toolkit GUI Components

The components of the Sun Java™ Wireless Toolkit for CLDC can all be launched from the command line. Each component is in the toolkit's `bin` directory.

TABLE B-1 Toolkit Component Commands

Command	Description
`DefaultDevice`	Displays a dialog that allows you to choose the default emulator skin
`ktoolbar`	Launches the user interface.
`prefs`	Launches the toolkit preferences
`utils`	Launches the toolkit utilities window

B.4 Setting Emulator Preferences

You can change the emulator preferences from the command line by using the `-Xprefs` option for the `emulator` command. The format is as follows:

`-Xprefs:`*filename*

Provide a *filename* that is the full path name of a property file whose values override the values in the preferences dialog box. The property file can contain the properties described in the following table.

TABLE B-2 Emulator Preferences Properties List

Property Name	Property Description and Legal Values
`http.version`	Network Configuration > HTTP Version Value: HTTP/1.1 | HTTP/1.0
`http.proxyHost`	Network Configuration > HTTP Address Value: hostname
`http.proxyPort`	Network Configuration > HTTP Port Value: integer
`https.proxyHost`	Network Configuration > HTTPS Address Value: hostname

TABLE B-2 Emulator Preferences Properties List *(Continued)*

Property Name	Property Description and Legal Values
`https.proxyPort`	Network Configuration > HTTPS Port Value: integer
`kvem.memory.monitor.enable`	Monitor > Enable memory monitor Value: true | false
`kvem.netmon.comm.enable`	Monitor > Enable Comm monitoring Value: true | false
`kvem.netmon.datagram.enable`	Monitor > Enable Datagram monitoring Value: true | false
`kvem.netmon.http.enable`	Monitor > Enable HTTP monitoring Value: true | false
`kvem.netmon.https.enable`	Monitor > Enable HTTPS monitoring Value: true | false
`kvem.netmon.socket.enable`	Monitor > Enable Socket monitoring Value: true | false
`kvem.netmon.ssl.enable`	Monitor > Enable SSL monitoring Value: true | false
`kvem.profiler.enable`	Monitor > Enable profiling Value: true | false
`netspeed.bitpersecond`	Performance > bits/sec combo box Value: integer
`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*)

TABLE B-2 Emulator Preferences Properties List *(Continued)*

Property Name	Property Description and Legal Values
`storage.size`	Storage > Storage size Value: integer
`mm.control.capture`	MMedia > Audio Capture Value: true | false
`mm.control.midi`	MMedia > MIDI tones Value: true | false
`mm.control.mixing`	MMedia > Audio Mixing Value: true | false
`mm.control.record`	MMedia > Audio Record Value: true | false
`mm.control.volume`	Value: true | false
`mm.format.midi`	MMedia > MIDI format Value: true | false
`mm.format.video`	MMedia > Video format Value: true | false
`mm.format.wav`	MMedia > WAV Audio format Value: true | false
`wma.client.phoneNumber`	WMA > Phone Number of Next Emulator Value: integer
`wma.server.firstAssignedPhoneNumber`	WMA > First Assigned Phone Number Value: integer
`wma.server.percentFragmentLoss`	WMA > % Random Message Fragment Loss Value: integer
`wma.server.deliveryDelayMS`	WMA > Message Fragment Delivery Delay (ms) Value: integer

B.5 Using Security Features

The full spectrum of the Sun Java™ Wireless Toolkit for CLDC's security features are also available from the command line. You can adjust the emulator's default protection domain, sign MIDlet suites, and manage certificates.

B.5.1 Changing the Emulator's Default Protection Domain

To adjust the emulator's default protection domain, use the following option with the `emulator` command:

`-Xdomain` *domain-type*

Assigns a security domain to the MIDlet suite. Domain types include `untrusted`, `trusted`, `minimum`, and `maximum`.

B.5.2 Signing MIDlet Suites

`JadTool` is a command-line interface for signing MIDlet suites using public key cryptography according to the MIDP 2.0 specification. Signing a MIDlet suite is the process of adding the signer certificates and the digital signature of the JAR file to a JAD file. `JadTool` is also capable of signing payment update (JPP) files.

`JadTool` only uses certificates and keys from Java SE platform keystores. Java SE software provides `keytool`, the command-line tool to manage Java SE platform keystores.

`JadTool` is packaged in a JAR file. To run it, open a command prompt, change the current directory to *toolkit*`\bin`, and enter the following command:

`java -jar JadTool.jar` *command*

The commands are as follows:

- `-help`

  Prints the usage instructions for `JADTool`.

- `-addcert -alias` *alias* `[-keystore` *keystore*`] [-storepass` *password*`] [-certnum` *number*`] [-chainnum` *number*`] [-encoding` *encoding*`] -inputjad | inputjpp` *input-file* `-outputjad | outputjpp` *output-file*

  Adds the certificate of the key pair from the given keystore to the JAD file or JPP file.

- `-addjarsig -jarfile` *jarfile* `-keystore` *keystore* `-alias` *alias* `-storepass` *password* `-keypass` *password* `-inputjad` *input-jadfile* `-outputjad` *output-jadfile*

  Adds the digital signature of the given JAR file to the specified JAD file. The default value for `-jarfile` is the `MIDlet-Jar-URL` property in the JAD file.

- `-showcert [([-certnum` *number*`] [-chainnum` *number*`]) |-all [-encoding` *encoding*`] -inputjad` *filename*

  Displays the list of certificates in the given JAD file.

- `-addjppsig -alias` *alias* `-keypass` *password* [`-keystore` *keystore*] [`-storepass` *password*] [`-encoding` *encoding*] `-inputjpp` *filename* `-outputjpp` *filename*

  Adds a digital signature of the input JPP file to the specified output JPP file.

The default value are as follows:

- `-encoding` - `UTF-8`
- `-jarfile` - `MIDlet-Jar-URL` property in the JAD file
- `-keystore` - `%HOMEPATH%\.keystore`
- `-certnum` - `1`
- `-chainnum` - `1`

B.5.3 Managing Certificates

`MEKeyTool` manages the public keys of certificate authorities (CAs), making it functionally similar to the `keytool` utility that comes with the Java SE SDK. 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 keystore. You can create one using the Java SE `keytool` utility.

Windows

`http://java.sun.com/javase/6/docs/technotes/tools/windows/keytool.html`

Linux

`http://java.sun.com/javase/6/docs/technotes/tools/solaris/keytool.html`

To run `MEKeyTool`, open a command prompt, change the current directory to *toolkit*`\bin`, and enter the following command:

Windows:	*toolkit*`\bin\mekeytool.exe` *-command*
Linux:	*toolkit*`/bin/mekeytool` *-command*

The command keywords follow. Note that while `MEKeyTool` runs from the installation directory, the default keys and keys you create will reside in your personal directory, *workdir*`\appdb`.

- `-help`

  Prints the usage instructions for MEKeyTool.

- `-import -alias` *alias* `[-keystore` *JCEkeystore*`] [-storepass` *storepass*`] -domain` *domain-name*

  Imports a public key into the ME keystore from the given JCE keystore using the given Java Cryptography Extension keystore password. The default ME keystore is *workdir*`\appdb\_main.mks` and the default Java Cryptography Extension keystore is *user.home*`\.keystore`.

- `-list`

  Lists the keys in the ME keystore, including the owner and validity period for each. The ME keystore is *workdir*`\appdb\_main.mks`.

- `-delete (-owner` *owner* `|` `-number` *key-number*`)`

  Deletes a key from the given ME keystore with the given owner. The ME keystore is *workdir*`\appdb\_main.mks`.

Note – The Sun Java™ Wireless Toolkit for CLDC contains an ME keystore called `_main.mks`, which is located in the `appdb` subdirectory. This keystore includes all the certificates that exist in the default Java SE platform keystore, which comes with the Java SE SDK installation.

B.6 Using the Stub Generator

Mobile clients can use the Stub Generator to access web services. The `wscompile` tool generates stubs, ties, serializers, and WSDL files used in Java API for XML (JAX) RPC clients and services. The tool reads a configuration file, which specifies either a WSDL file, a model file, or a compiled service endpoint interface. The syntax for the stub generator command is as follows:

`wscompile [`*options*`]` *configuration-files*

B.6.1 Options

TABLE B-3 Options for the `wscompile` Command

Option	Description
`-d` *output directory*	Specifies where to place generated output files
`-f:`*features*	Enables the given features
`-features:`*features*	Same as `-f:`*features*

TABLE B-3 Options for the `wscompile` Command *(Continued)*

Option	Description
`-g`	Generates debugging info
`-gen`	Same as `-gen:client`
`-gen:client`	Generates client artifacts (stubs, etc.)
`-httpproxy:`*host:port*	Specifies a HTTP proxy server (port defaults to 8080)
`-import`	Generates interfaces and value types only
`-model` *file*	Writes the internal model to the given file
`-O`	Optimizes generated code
`-s` *directory*	Specifies where to place generated source files
`-verbose`	Outputs messages about what the compiler is doing
`-version`	Prints version information
`-cldc1.0`	Sets the CLDC version to 1.0 (default). Float and double become String.
`-cldc1.1`	Sets the CLDC version to 1.1 (float and double are OK)
`-cldc1.0info`	Shows all CLDC 1.0 information and warning messages.

Note – Exactly one `-gen` option must be specified. The `-f` option requires a comma-separated list of features.

TABLE B-4 lists the features (delimited by commas) that can follow the `-f` option. The `wscompile` tool reads a WSDL file, compiled service endpoint interface (SEI), or model file as input. The Type of File column indicates which of these files can be used with a particular feature.

TABLE B-4 Command Supported Features (`-f`) for `wscompile`

Option	Description	Type of File
`explicitcontext`	Turns on explicit service context mapping	WSDL
`nodatabinding`	Turns off data binding for literal encoding	WSDL
`noencodedtypes`	Turns off encoding type information	WSDL, SEI, model
`nomultirefs`	Turns off support for multiple references	WSDL, SEI, model
`novalidation`	Turns off full validation of imported WSDL documents	WSDL
`searchschema`	Searches schema aggressively for subtypes	WSDL

TABLE B-4 Command Supported Features (`-f`) for `wscompile`

Option	Description	Type of File
`serializeinterfaces`	Turns on direct serialization of interface types	WSDL, SEI, model
`wsi`	Enables WSI-Basic Profile features (default)	
`resolveidref`	Resolves `xsd:IDREF`	
`nounwrap`	No unwrap.	

Example

```
wscompile -gen -d generated config.xml
wscompile -gen -f:nounwrap -O -cldc1.1 -d generated config.xml
```

APPENDIX **C**

Localization

This appendix describes setting the language displayed in the Sun Java™ Wireless Toolkit for CLDC and the localization setting of the emulation environment.

C.1 Locale Setting

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

Some software uses a locale to tailor information for users, such as:

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

By default, all strings in the user interface are displayed in the language of the supported platform's locale.

For example, Japanese characters can be displayed in a toolkit that is running on a Japanese Microsoft Windows machine, provided that the correct localized Sun Java™ Wireless Toolkit for CLDC is downloaded and installed.

You can set the `wtk.locale` property to have the user interface displayed in a specified locale's language. As discussed in Section 3.8, "Configuring the Wireless Toolkit" on page 3-14, you must copy `ktools.properties` from the installation directory to your working directory (*workdir*`/wtklib/ktools.properties`), then edit your copy. For example, you can have the toolkit running on a Japanese machine but still have the user interface 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 Sun Java™ Wireless Toolkit for CLDC.

C.2 Emulated Locale

A device's locale is contained in the system property `microedition.locale`. You can change the emulator's locale by choosing Edit > Preferences and selecting i18n. Choose a locale from the combo box or type it in directly.

For information on `microedition.locale`, consult the MIDP specification.

C.3 Character Encodings

The CLDC system property, `microedition.encoding`, defines the default character encoding name of the MIDP environment. In the Sun Java™ Wireless Toolkit for CLDC emulator, 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 Java SE platform running on the same window system. For example, in an English window system, the encoding setting is as follows:

```
microedition.encoding=ISO8859_1
```

You can override the default value by adding the `microedition.encoding` property to the *workdir*`\wtklib\ktools.properties` file. For example, if you want to use UTF-8 as the default setting on Microsoft Windows, you can set the property in the *workdir*`\wtklib\ktools.properties` file as follows:

```
microedition.encoding=UTF-8
```

For more information on character encoding, see the CLDC specification.

Note – All the Java SE platform encoders are available in the emulated environment. See the *Sun Java™* Wireless Toolkit for CLDC *Basic Customization Guide* for information on how to limit the list of available encoders for a specific device.

C.4 Java Technology 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 Java SE platform running on the same window system.

You can override the default value by adding the `javac.encoding` property to the `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
```

C.5 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 Java SE platform Java technology 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 *Sun Java™ Wireless Toolkit for CLDC Basic Customization Guide* for information on how to configure them.

Index

N

O

P

R

S

T

U

V

W

X