Oracle® Java Micro Edition Software Development Kit

Developer's Guide

Release 3.2 for Eclipse on Windows

E37550-02

October 2012

This document describes how to use the Java ME SDK plugin for Eclipse on Windows.

ORACLE®

Oracle Java Micro Edition Software Development Kit, Release 3.2 for Windows

E37550-02

Contents

Preface

The Oracle® Java ME SDK is mobile application development tool available as a plugin to the NetBeans IDE and the Eclipse IDE.

Audience

This document is intended for Java ME application developers.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/us/corporate/accessibility/index.html`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/support/contact.html` or visit `http://www.oracle.com/accessibility/support.html` if you are hearing impaired.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Related Documents

For more information, see the following documents:

- To see documentation for the Oracle Java Wireless Client and CLDC Hotspot go to `http://download.oracle.com/javame/mobile.html` and look under Documentation for Device Makers

- For SDK, LWUIT, and legacy Sun Java Wireless Toolkit documentation see `http://download.oracle.com/javame/developer.html`.

1 Getting Started

The Oracle® Java Micro Edition (Java ME) Software Development Kit (SDK) is a natural starting point for learning and using Java ME technology. The focus of the SDK is to provide emulation and deployment assistance during the development process. This chapter introduces the SDK and provides a quick introduction to using the SDK.

Using this simple yet powerful tool you can create, edit, compile, package, and sign an application. After testing your application in the Oracle Java ME SDK emulation environment, you can move to deploying and debugging on a real device.

This SDK provides supporting tools and sample implementations for the latest in Java ME technology. The SDK provides support for recent versions of the Connected Limited Device Configuration (CLDC) and Information Module Profile - Next Generation (IMP-NG).

- Section 1.1, "Quick Start"
- Section 1.2, "Java ME SDK Update Center"

As of version 3.2, the Java ME SDK is a plugin to the Eclipse IDE. Please see the Release Notes at:
`http://docs.oracle.com/javame/dev-tools/jme-sdk-3.2/release-notes/toc.htm`.

1.1 Quick Start

The Oracle Java ME SDK plugin uses Eclipse technology, as described in the Eclipse online help. These tips offer some hints for getting started as quickly as possible.

- Access the documentation. The online help is the primary documentation for the SDK. Many windows and dialogs feature a help button that opens context-sensitive help in the help viewer. You can also type F1.

  Select Help > Help Contents to open the Eclipse Online Help viewer. Remember to use the search capability to help you find topics.

  > **Note:** If you require a larger font size, the help topics are also available as a printable PDF and a set of HTML files.

- Run sample projects. Running sample projects is a good way to become familiar with the SDK.

  See Section 3.2, "Running a Project" for a general overview of how to run a project.

1.1.1 Verify Your Installation

The Oracle Java ME SDK has two components.

- The Oracle Java ME SDK platform software installation includes the supported runtimes, emulation software, device configurations, supporting libraries, documentation (including Javadocs), the command line interface, and other utility executables. The default location of the SDK, as specified by its installer, is:

  ```
  C:\Java_ME_platform_SDK_3.2
  ```

- The Oracle Java ME SDK plugins for Eclipse add Oracle Java ME SDK technology to an Eclipse IDE installation running Mobile Tools for Java (MTJ). The outstanding feature of Oracle Java ME SDK is device emulation for the Connected Device Limited Configuration (CLDC). If the plugin is properly installed you will see the Device Selector tab on the bottom left, or, if it is not visible you can select Window > Show View > Device Selector.

  The SDK provides two unique instances for most devices. For example, JavaMEPhone1 and JavaMEPhone2 are the same except for the device number and the phone number, so you can perform tests that require two devices (messaging, for example) without customization.

  See Chapter 5, "Working With Devices."

- If the plugin is installed, and you cannot see the device selector you can add the Java ME SDK devices manually.

  Select Window > Open Perspective > Other > Java ME and click OK.

  Go to Window > Preferences > Java ME > Device Management and click on Manual Install.

  In the Specify Search Directory field, insert the path to the Java ME SDK platform installation directory, and press Enter. The devices appear on the Devices table. Click on Finish and then click on OK. The device selector opens.

1.1.2 Project Quick Start

- See the Package Explorer window and the Navigator window for a visual overview of the logical and physical layout of a project. When viewing items in the tree, use the context menu (right-click) to see the available actions.

- A project has a default device that is used when it is run from the toolbar by clicking the green arrow or going to the Run menu and choosing Run.

  To see a project's default device, expand the project node and double-click the Application Descriptor subnode. The project's Overview tab opens in the central editing area. The Runtime table on the bottom right of the Application Descriptor panel lists devices (execution environments) for the current project. The checked device will be used when the project is run. You can easily add, modify, or remove a device.

- To run an application on different devices without changing the default device, right-click a device in the Device Selector pane, then select Run Project and choose an open project.

- The emulator is an independent process, and when it has started it is a separate process from the build process running in Eclipse. Stopping the build process or closing a project does not always affect the application running in the emulator. You must be sure to terminate the application (the emulator can remain open). See Section 3.2, "Running a Project."

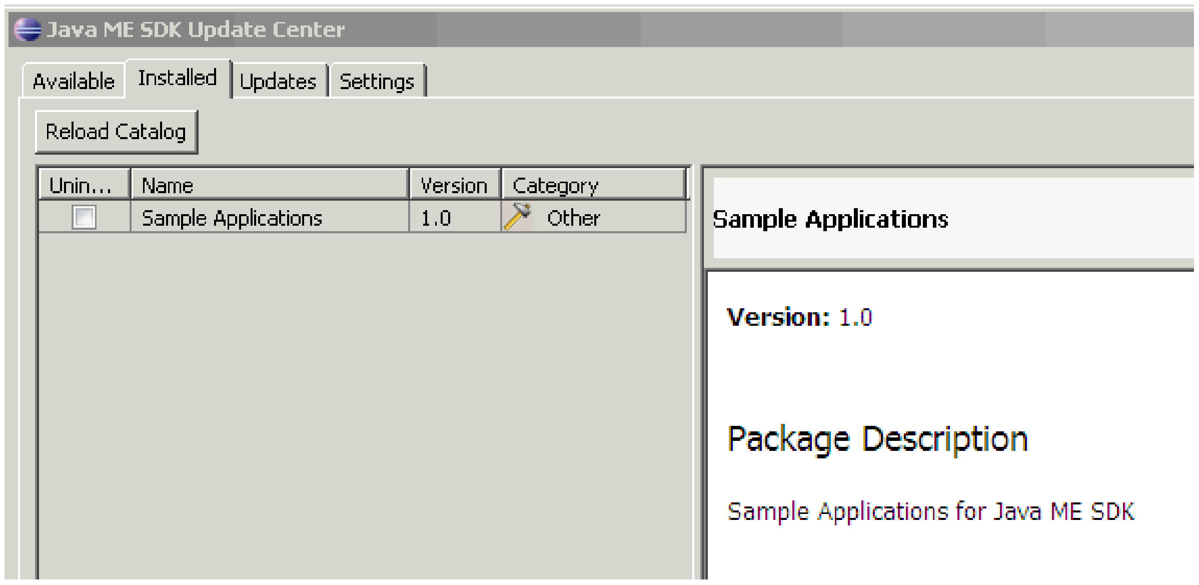
1.2 Java ME SDK Update Center

The Java ME SDK Update Center supports automatic updating of the entire Java ME SDK plugin, and individual modules within the Java ME SDK. To access the update center, select Help > Java ME SDK Update Center. The update manager works separately from the Eclipse Check for Updates function so that the Java ME SDK plugin components, including the Update Center, can be updated independently.

Oracle Java ME SDK is delivered as two plugins in a category named Java ME SDK Tools. The plugins are:

- Java ME SDK Tools
- Java ME SDK Demos

To detect updates, select Help > Java ME SDK Update Center and choose the Available tab. Any available updates are listed. Choose an update and click Install to update the plugin. The plugins then appear as activated on the Installed tab.



Unin...	Name	Version	Category
☐	Sample Applications	1.0	Other

Sample Applications

Version: 1.0

Package Description

Sample Applications for Java ME SDK

Demos are delivered separately for two reasons:

- Some demos use network access for test purposes, however, the sample code does not include protection against malicious intrusion. Before using the demos, see the "Installation and Runtime Security Guidelines" in the *Oracle Java Micro Edition Software Development Kit Release Notes*.
- Sample code has a different copyright that allows you to redistribute provided the Oracle copyright is kept.

For more on sample applications, see Chapter 3, "Using Sample Projects"

2 Platforms

This chapter describes the Oracle Java ME SDK technology platforms, also called stacks. They are: CLDC with MIDP and IMP-NG, as discussed in Section 2.1, "Emulation Platforms."

A project runs on a particular emulation platform. The device manager determines whether a device is appropriate for your project based on the platform, the APIs your application uses, and a set of device properties. If you run an application and an appropriate emulator or device is currently running, the SDK automatically installs and runs your application in the current device so that you do not have to launch the emulator repeatedly.

2.1 Emulation Platforms

An emulator simulates the execution of an application on one or more target devices. An emulation platform enables you to understand the user experience for an application and test basic portability. For example, a platform enables you to run applications on several sample devices with different features, such as screen size, keyboard, runtime profile and other characteristics.

Oracle Java ME SDK provides the following emulation platforms:

- CLDC with Mobile Information Device Profile (MIDP)
- CLDC with MIDP, Information Module Profile - Next Generation (IMP-NG) subset

All platforms include predefined devices with different screen sizes, runtime profiles, and input methods.

See Section 2.1.1, "CLDC with MIDP" and Section 2.1.2, "IMP-NG."

2.1.1 CLDC with MIDP

CLDC/MIDP applications conform to both the Connected Limited Device Configuration and the Mobile Information Device Profile (`http://jcp.org/en/jsr/detail?id=139`). The CLDC/MIDP stack supports the following technology.

- CLDC 1.1 and MIDP 2.1
- All the JSRs listed in Table 13.1.

CLDC/MIDP applications are targeted for devices that typically have the following capabilities:

- A 16-bit or 32-bit processor with a clock speed of 16MHz or higher

- At least 160 KB of non-volatile memory allocated for the CLDC libraries and virtual machine
- At least 192 KB of total memory available for the Java platform
- Low power consumption, often operating on battery power
- Connectivity to some kind of network, often with a wireless, intermittent connection and limited bandwidth

Typical devices might be cellular phones, pagers, low-end personal organizers, and machine-to-machine equipment. In addition, CLDC can also be deployed in home appliances, TV set-top boxes, and point-of-sale terminals.

The SDK provides two default emulators to support CLDC:

- ClamshellJavaMEPhone1

  A flip phone with a primary display and a secondary display.
- JavaMEPhone1 and JavaMEPhone2

  A flat touch screen device.

These devices support CLDC 1.1, MIDP 2.1, and optional packages for JSRs 75, 82, 135, 172, 177, 179, 184, 205, 211, 226, 234, 239, 256, 257, and 280.

See Section 3.6, "Running MIDP and CLDC Sample Projects" and Chapter 5, "Working With Devices."

2.1.2 IMP-NG

JSR 228 describes the *Information Module Profile - Next Generation*, referred to as IMP-NG. This JSR extends and enhances *JSR 195: Information Module Profile*.

The IMP-NG implementation depends upon CLDC 1.0. It is a strict subset of MIDP 2.0 that excludes MIDP 2.0 graphical display capabilities, resulting in a smaller footprint appropriate for Information Modules (IMs). Potential devices for CLDC with IMP-NG might be modems, home electronics devices, or industrial metering devices.

An IMP-NG application is an IMlet, and multiple IMlets in a single JAR file form an IMlet suite. When creating an IMlet project you follow the same process that you use to create a Java ME Mobile Application project and select an IMP-NG device. The device selection determines the supported JSRs.

The IMP-NG stack supports the following JCP APIs: JSRs 75, 120, 172, 177, 179, 257, and 280. In addition, Oracle provides APIs to support IMP-NG development, as described in Section 13.2, "Oracle APIs."

The Java ME SDK implementation provides IMP-NG emulation, on-device tooling connectivity to real devices, and Attention (AT) Command support. The SDK emulator supports IMP-NG with IMPNGPhone1 and IMPNGPhone2 skins and provides simple interfaces for Inter-Integrated Circuit (I^2C), Serial Peripheral Interface (SPI), General Purpose Input/Output (GPIO), and Memory-mapped I/O (MMIO) buses. The emulator's external event generator provides a way for you to inject calls to emulate AT Commands, alter basic pin and port information for GPIO, and memory block values.

See Section 3.7, "Running IMP-NG Sample Projects."

3 Using Sample Projects

The Oracle Java ME SDK sample projects introduce you to the emulator's API features and the SDK features, tools, and utilities that support the various APIs. These features can help you customize the sample projects or create applications of your own.

Note: As mentioned in Section 1.2, "Java ME SDK Update Center" the demos are delivered and installed separately. Before using the demos, please see the "Installation and Runtime Security Guidelines" in the *Oracle Java Micro Edition Software Development Kit Release Notes*. Some demos use network access and open ports. Because the sample code does not include protection against malicious intrusion, you must ensure your environment is secure should you choose to run the sample projects.

For instructions on running projects, see the following topics:

- Section 3.1, "Creating a Sample Project"
- Section 3.2, "Running a Project"
- Section 3.3, "Troubleshooting"
- Section 3.4, "Sample Project Overview"
- Section 3.5, "Configuring the Web Browser and Proxy Settings"
- Section 3.6, "Running MIDP and CLDC Sample Projects"
- Section 3.7, "Running IMP-NG Sample Projects"

3.1 Creating a Sample Project

Sample applications are installed in a separate Eclipse plugin. Do not run or edit these projects directly. You create a new project that is an instance of the sample project.

The default location for Oracle Java ME SDK projects is the default Eclipse project directory. Each project has a `src` directory that contains Java programming language source code. For example, the default location of the source code for the SMS sender MIDlet (`example.sms.SMSSend`) in `WMADemo` resides in the following location:

EclipseWorkspace`\WMADemo\src\example\sms\SMSSend.java`

1. Go to File > New > Project and in the Categories window select Examples > Java ME SDK 3.2 and click on Java ME Sample Applications. Click Next. In the next screen choose one of the samples and click Finish.

2. If prompted, choose an emulator platform and a device. Note, changing the device affects the possible device profiles, so you might need to explicitly select a profile. Click OK.

   The project is added to the Package Explorer window.

   > **Note:** If you can't see the Packager Explorer window choose Window > Show View > Package Explorer. To see console output, select Window > Show View > Console.

3.2 Running a Project

Create your own project, or instantiate one of the sample projects provided with the SDK as described in Section 3.1, "Creating a Sample Project."

1. Use one of these methods to run a project.

   - Select a project and click the green Run button in the toolbar.
   - Right-click a project and select Run As > Emulated Java ME JAD from the context menu.
   - To run a project on a different device, or to change the execution mode, choose the device in the Device Selector window (Window > Show View > Other > Java ME SDK Tools > Device Selector).

     Right-click on a device and select Run Project from the context menu. Pull right to see a listing of open projects.

   The device emulator window opens with the demo application running. If the demo is a MIDlet suite you might have to choose a MIDlet to launch.

2. As the sample project runs, soft keys might be enabled below the screen on the left or right side.

   You use soft keys to install or launch an application, open a menu, exit, or perform some other action. Some demos include these instructions in the application.

   For instructions on running samples, see Table 3–1.

3. When you are finished viewing an application, go to the emulator's Application menu and select Exit to close the emulator and stop the execution of the project's build script.

   When the emulator is launched, it runs as an independent process. Stopping the application from Eclipse does not close the emulator instance.

   Likewise, closing the Eclipse IDE does not affect the emulator instance. In the emulator, select Application > Exit or press the emulator’s exit button (the X) on the upper right to ensure that both the emulator process and the project build process close.

3.3 Troubleshooting

Sometimes even a "known good" application, such as a sample project, does not run successfully. The problem is usually your environment.

- Some demonstrations require specific setup and instructions. For example, if a sample uses web services and you are behind a firewall, you must configure the emulator's proxy server settings or web access fails. See Section 3.5, "Configuring the Web Browser and Proxy Settings."

- If an application must run over the air (OTA), the SDK automatically installs it in the device instance. See Section 11.3, "Emulator Command Line Options."

  MIDlet Suites use `runMIDlet` to perform the installation.

  installdir`\runtimes\cldc-hi\bin\runMidlet.exe`

  Because these programs are launched remotely, virus checking software can prevent them from running. If this happens, the project compiles, but the emulator never opens. In the console you see warnings that the emulator cannot connect.

  Consider configuring your antivirus software to exclude `runMidlet` from checking.

3.4 Sample Project Overview

The Oracle Java ME SDK includes demonstration applications that highlight some technologies and APIs that are supported by the emulator.

Most demonstration applications are simple to run. Section 3.2, "Running a Project" contains instructions for running most demonstrations. Sample projects usually have some additional operation instructions.

Table 3–1 lists all the MIDP/CLDC demonstration applications that are included in this release.

Table 3–1 MIDP/CLDC Sample Projects

Sample	Optional Package	Description	Instructions
Advanced Multimedia Supplements	JSR 234	Demonstrates 3D audio, reverberation, image processing, and camera control.	Section 3.6.1, "Running the AdvancedMultimediaSupplements Sample Project"
AudioDemo	MMAPI 1.1	Demonstrates audio capabilities, including mixing and playing audio with an animation.	Section 16.4, "Running AudioDemo"
BluetoothDemo	JSR 82	Demonstrates device discovery and data exchange using Bluetooth.	Section 15.2, "Running the Bluetooth Demo"
CHAPIDemo	JSR 211	A content viewer that also uses MediaHandler.	Chapter 21, "JSR 211: Content Handler API (CHAPI)"
CityGuide	JSR 179	A city map that displays landmarks based on the current location.	Section 18.2, "Running the CityGuide Sample Project"
Contactless	JSR 257	Emulates detection of RFID tags.	Section 25.1, "Using ContactlessDemo"
Demo3D	JSR 184	Contains MIDlets that demonstrate how to use 3D graphics, both immediate mode and retained mode.	Section 20.4, "Running Demo3D Samples"
Demos	MIDP 2.0	Includes various examples: animation, color, networking, finance, and others.	Section 3.6.2, "Running the Demos Sample Project"
FPDemo	CLDC 1.1	Simple floating point calculator.	Section 3.6.3, "Running FPDemo"
Games	MIDP 2.0	Includes TilePuzzle, WormGame, and PushPuzzle.	Section 3.6.4, "Running Games".
LWUITBrowser	N/A	Demonstrates LWUIT features.	Chapter 9, "Lightweight UI Toolkit"
LWUITDemo	N/A	Demonstrates LWUIT features.	Chapter 9, "Lightweight UI Toolkit"

Table 3–1 (Cont.) MIDP/CLDC Sample Projects

Sample	Optional Package	Description	Instructions
LWUITIODemo	N/A	Demonstrates LWUIT features.	Chapter 9, "Lightweight UI Toolkit"
LWUITMakeover	N/A	Demonstrates LWUIT features.	Chapter 9, "Lightweight UI Toolkit"
LWUITSpeed	N/A	Demonstrates LWUIT features.	Chapter 9, "Lightweight UI Toolkit"
LWUITTimeZone	N/A	Demonstrates LWUIT features.	Chapter 9, "Lightweight UI Toolkit"
LWUITTipster	N/A	Demonstrates LWUIT features.	Chapter 9, "Lightweight UI Toolkit"
MMAPIDemos	MMAPI	Demonstrates MMAPI features, including tone sequences, MIDI playback, sampled audio playback, and video.	Section 16.5, "Running MMAPIDemos"
Multimedia	MMAPI	Demonstrates different video playback formats.	Section 16.5.3, "Video"
NetworkDemo	MIDP 2.0	Demonstrates how to use datagrams and serial connections.	Section 3.6.5.1, "Socket Demo" and Section 3.6.5.2, "Datagram Demo"
ObexDemo	JSR 82	Demonstrates device discovery and data exchange using Bluetooth.	Section 15.3, "Running the OBEX Demo"
PDAPDemo	JSR 75	Demonstrates how to manipulate contacts, calendar items, and to-do items. Demonstrates accessing local files.	Section 14.3, "Running PDAPDemo"
PhotoAlbum	MIDP 2.0	Demonstrates a variety of image formats.	Section 3.6.6, "Running PhotoAlbum"
SATSADemos	JSR 177	Demonstrates communication with a smart card and other features of SATSA.	Section 17.4, "Running SATSADemos"
SATSAJCRMIDemo	JSR 177	Shows how to use the SATSA-Java Card Remote Invocation method.	Section 17.5, "Running SATSAJCRMIDemo"
Sensors	JSR 256	The SensorBrowser and Marbles game demonstrate sensor input.	Section 24.4, "SensorBrowser" and Section 24.5, "Marbles"
SVGContactList	JSR 226	Uses SVG to create a contact list displayed with different skins.	Section 22.2, "Running SVGContactList"
SVGDemo	JSR 226	Uses different SVG rendering techniques.	Section 22.1, "Running SVGDemo"
UIDemo	MIDP 2.0	Showcases the breadth of MIDP 2.0's user interface capabilities.	Section 3.6.7, "Running UIDemo"
WMADemo	WMA 2.0	Shows how to send and receive SMS, CBS, and MMS messages.	Section 19.2, "Running WMADemo"
XMLAPIDemo	JSR 280	Uses DOM and StAX APIs to create an XML sample and SAX, DOM and StAX APIs to parse the sample.	Follow the instructions the application provides.

3.5 Configuring the Web Browser and Proxy Settings

If you are behind a firewall you must configure the proxy server so that MIDP applications using web services can succeed.

The settings are typically the same as those you are using in your web browser. Select Window > Preferences > General > Network Connections and change the settings as appropriate.

3.6 Running MIDP and CLDC Sample Projects

This topic gathers MIDP and CLDC samples that are not discussed in separate chapters. This is the case when a sample uses many JSRs, or when a supported JSR does not have any special implementation details.

- Section 3.6.1, "Running the AdvancedMultimediaSupplements Sample Project"
- Section 3.6.2, "Running the Demos Sample Project"
- Section 3.6.3, "Running FPDemo"
- Section 3.6.4, "Running Games"
- Section 3.6.5, "Running Network Demo"
- Section 3.6.6, "Running PhotoAlbum"
- Section 3.6.7, "Running UIDemo"

For other CLDC demos, see Table 3–1.

3.6.1 Running the AdvancedMultimediaSupplements Sample Project

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

Section 3.6.1.1, "Image Effects"
Section 3.6.1.2, "Music Effects"
Section 3.6.1.3, "Camera"
Section 3.6.1.4, "Moving Helicopter"

3.6.1.1 Image Effects

This MIDlet demonstrates standard image processing operations.

- Launch the Image Effects MIDlet.
- Choose input and output image formats, and press Done. The input image and output images are displayed simultaneously.
- Choose an effect from the Menu and click the Done button to apply a transformation, effect or overlay. The source image is shown above and the processed image is shown below. Some items, Set Transforms, for example, can perform several operations in a single transaction.

  The menu options are as follows:

  - Reset - Set transforms, effects, and overlays to the initial state.
  - Monochrome Effects - Activate grayscale rendering.
  - Negative Effect -Reverse dark and light areas.
  - Set Formats - Select an input object type and an output image format.
  - Set Effect Order - Specify the order in which transforms, effects and overlays are applied.
  - Set Transforms - Change width and height scale, border, and rotation options.

- Set Overlays - Specify the color and orientation of a color block overlay.

3.6.1.2 Music Effects

Demonstrates the advanced audio capabilities of the Advanced Multimedia Supplements. As an audio file loops continuously, you can adjust the volume, and reverberation settings.

3.6.1.3 Camera

This MIDlet demonstrates 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.

- Zoom settings - digital and optical zoom settings 100-300 in increments of 20. Make a selection and press Back.
- View gallery - View a list of the snapshots stored in: *userhome*`\javame-sdk\3.2\work\`*device*`\appdb\filesystem\root1`. Choose Display to see the snapshot. You have the option to delete the file from disk. If `\root1` is empty use the Snapshot option to save images for the gallery.
- Set flash mode - Off, AUTO, AUTO_WITH_REDEYEREDUCE, FORCE, FORCE_WITH_REDEYEREDUCE, FILLIN.
- Change F_Stop number - 0, 400, 560, 800, 1600.
- Choose exposure modes - Preset modes are auto, landscape, snow, beach, sunset, night, fireworks, portrait, backlight, spotlight, sports, text.
- Disable/Enable shutter feedback.
- Exit - Close this MIDlet and return to the initial window.
- Snapshot setting - Set whether to display the snapshot on the screen or print it to a file. Snapshots are stored in:

  userhome`\javame-sdk\3.2\work\`*emulator_name*`\appdb\filesystem\root1`

3.6.1.4 Moving Helicopter

Simulates a helicopter (red dot) flying around a stationary observer (blue dot). Use headphones for best results. You can control the parameters of the simulation with the soft menu options: Volume, Location settings, Spectator orientation, and Distance Attenuation settings. After viewing menu options, press the close button (the X on the right) to return to the helicopter scenario.

With the Location settings be aware that supplying large values for the screen width or flight altitude means the helicopter might be out of range - that is, it flies off the screen and you might not be able to hear it.

For spectator orientation stereo headphones or speakers help detect the difference in position, assuming your volume and location settings put the helicopter in audible range. The same is true for the Distance Attenuation settings, which enable you to control the doppler effect.

3.6.2 Running the Demos Sample Project

This demo contains several MIDlets that highlight different MIDP features. Click or use the navigation keys to highlight a MIDlet, then choose the Launch soft key.

- Section 3.6.2.1, "Colors"

- Section 3.6.2.2, "Properties"
- Section 3.6.2.3, "Http"
- Section 3.6.2.4, "FontTestlet"
- Section 3.6.2.5, "Stock"
- Section 3.6.2.6, "Tickets"
- Section 3.6.2.7, "ManyBalls"
- Section 3.6.2.8, "MiniColor"
- Section 3.6.2.9, "Chooser"
- Section 3.6.2.10, "HttpExample"
- Section 3.6.2.11, "HttpView"
- Section 3.6.2.12, "PushExample"

3.6.2.1 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 enables you to jump in increments of 4 (Fine) or 32 (coarse).
- You can change the color on any or all of the vertical bars.

3.6.2.2 Properties

This MIDlet displays your system property values. The output is similar to the following values:

```
Free Memory = 2333444
Total Memory = 4194304
microedition.configuration = "CLDC-1.1"
microedition.profiles = "MIDP-2.1"
microedition.platform = "generic"
microedition.platform = "en-US"
microedition.platform = "IS08859_1"
```

3.6.2.3 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.

- Right-click on Demos and select Properties.

  - Select the Running category.
  - Select Regular Execution.
  - Check Specify the Security Domain and select Maximum.
  - Click OK.
- If you are using a proxy server, you must configure the emulator's proxy server settings as described in Section 3.5, "Configuring the Web Browser and Proxy Settings." The HTTP version must be 1.1.
- If you are running antivirus software it might be necessary to create a rule that allows your MIDlet to permit connections to and from a specific web site. See Section 3.3, "Troubleshooting."

Running the Demo

Launch the Http MIDlet. To test, choose the Menu soft key and choose Get, Post, or Head to test the selected URL.

Http Test returns the information it obtains. 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 to choose an action. The Menu items vary depending on the screen you are viewing.

- Choose `Qwerty` to set the input type. This activates a submenu with the options Qwerty, 123, Abc, Predict, and Symbols. This choice is present if you have the option to edit a URL (select Choose, then click the Add soft button).
- Choose `GET` or press the Get soft key to retrieve data from the selected URI.
- Choose `POST` to retrieve the post information from the server handling the selected page.
- Choose `HEAD` to retrieve only the META information from the headers for the selected URI.
- Select Choose to bring up the current list of web pages. You can chose a different page or add your own page to the list. To specify a new URL, choose the Add soft button. The screen displays `http://`. Type in the rest of the URL. If necessary select Qwerty on the menu and choose a different input method. Be 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.

3.6.2.4 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.

3.6.2.5 Stock

Like the Http demonstration, this sample uses an HTTP connection to obtain information. Use the same preparation steps as Section 3.6.2.3, "Http."

Run the Demos project and launch the Stock MIDlet.

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

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 `ORCL` 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 Settings > Updates to change the interval. Use the navigation arrows to select one of Continuous, 15 minutes, 30 minutes, one hour, or three 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.

Stock Tracker

Stock Tracker displays a list of the stocks you added and their current values. Click a stock to display additional information. For example, the last trade and the high and low values.

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.

Alerts

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

From the Stock Menu application home screen, select Alerts.

Select Add.

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

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

3.6.2.6 Tickets

This demonstrates how an online ticket auction application might behave. Click Done to continue to the Welcome To Tickets page. The Choose a Band field displays BootWare & Friends by default, and an auction ticker runs at the bottom of the screen.

Choose a band from the dropdown menu. The available auction appears.

Select Make a Bid from the menu. Use the arrow keys to move from field to field. Fill out each field, then select the Next soft key. The application asks you to confirm your bid. Press the Submit button or use the arrow keys to highlight Submit then press the Submit soft key. You receive a Confirmation number. Click Bands to return to the Bands page.

Select set an alert, select Set an Alert from the soft Menu. In the bid field type in a value higher than the current bid and continue bidding as before. 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 to the Choose a Band dropdown list, select the Menu soft key and choose Add Bands. Type in a band name or a comma-delimited list of names and choose Save from the soft menu. After a confirmation message, choose Done and you are returned to the Welcome To Tickets page. The added band(s) are displayed at the end of the Choose a Band drop-down menu. If you don't see your bands, use the down arrow key to move to the bottom of the list.

Note, this is only a demonstration. To fully describe the band you must edit the following file:

`src\example\auction\NewTicketAuction.java`.

To remove a band, select the Menu soft key and Remove Bands. Check a box for one or more 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 Show More Info. If desired, use the arrow keys and the select key to change these values. Choose the Save soft key.

3.6.2.7 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.

3.6.2.8 MiniColor

This MIDlet sets an RGB value. Use navigation keys to change color values.

Keyboard controls work as you would expect. First cursor up or down to highlight a color, and then use left and right keys to lower and raise the value of the selected color.

3.6.2.9 Chooser

The Chooser application uses a variety of controls to change text color, background color, and fonts.

- Choose Menu > Text Color. Change the color as described for MiniColor and select the OK soft button.
- Choose Menu > Background Color. Change the color as described for MiniColor and select the OK soft button.
- Choose Menu > Fonts. You can change the font Face, Style, and Size.

  Cursor up and down to highlight a property item by item. The left and right keys jump between lists. Click select to choose the highlighted option. The chosen option's circle or square is filled with white.

  Click OK to display a sample formatted with your settings.

  Click Exit to return to the home screen.

3.6.2.10 HttpExample

This sample makes an HTTP communication. A popup confirms the transaction was successful.

3.6.2.11 HttpView

This application displays three predefined URLs.

Choose a URL, and press the soft buttons to cycle through Head, Headers, Requests, and Errors.

Alternatively, Use the menu options.

3.6.2.12 PushExample

This application simulates a feed. As soon as you connect, you receive and display a graphic. Select Done to continue.

3.6.3 Running FPDemo

FPDemo is a simple floating point calculator.

1. Enter a number in the first field.
2. To choose an operator, highlight the drop-down list and click to select. Cursor down to highlight an operator, then click to make a selection.
3. Enter a second value.
4. From the Menu, select Calc to calculate the result.

3.6.4 Running Games

This application features three games: TilePuzzle, WormGame, and PushPuzzle.

TilePuzzle. The desired result, "Rate your mind pal" is shown first. From the soft Menu, select Start. The scrambled puzzle is displayed. The arrow keys move the empty space, displacing tiles accordingly. To change the arrow behavior, select Options from the soft menu, and choose reverse arrows, then click OK. From the menu you can Reset, or change options.

WormGame. From the soft Menu, select 1, Launch. Use the arrow keys to move the worm to the green box without touching the edge of the window. When the game is launched, use the soft menu to change game options.

PushPuzzle. Use the blue ball to push the orange boxes into the red squares in the fewest number of moves.

3.6.5 Running 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. For example, run the demo on JavaMEPhone1 and JavaMEPhone2.

3.6.5.1 Socket Demo

In this application one emulator acts as the socket server, and the other as the socket client.

1. In the first emulator, launch the application, then select the Server peer. Choose Start. The Socket Server displays a status message that it is waiting on port 5000.

2. In the second emulator, launch the application, select the Client peer, then choose Start. Choose Start to launch the client. The Socket Client displays a status message that indicates it is connected to the server on port 5000. 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. Choose Send from the menu. The server emulator activates a blue light when the message is received.

3. 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`. From the menu, select Send.

4. Back in the Socket Client, the status is: `Message received - Hello Client, I heard you`. Until you send a new message, the Send box contains the previous message you sent.

3.6.5.2 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.

1. In the first emulator, launch Datagram Demo, then select the Server peer. Choose Start. Initially, the Datagram Server status is `Waiting for connection on port 5555,` and the Send box is empty.

2. In the second emulator, launch Datagram Demo, select the Client peer, ensure the port number is 5555 and choose Start. The Datagram Client status is: `Connected to server on port 5555`. Use the down navigation arrow to highlight the Send box. Type a message in the Send box, then choose the Send from the menu. For example, type `Hello datagram server.` From the menu, select Send.

3. 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.

4. In the Datagram Client, the status field displays the message received from the server. The Send box contains the last message you sent. Overwrite it to send another message.

3.6.6 Running PhotoAlbum

The PhotoAlbum demo displays both static and animated images. When you are displaying an image, you can use the Options soft menu to change the borders. If the image is animated, you can change the speed of the playback.

3.6.7 Running UIDemo

UIDemo showcases a variety of MIDP user interface element implementations. Most elements have some interactive capability (navigate and select) and some allow keypad or keyboard input.

Input interaction is similar across demos. You can choose items from lists or type in data.

This demo implements three list selection methods:

- Exclusive (radio buttons)
- Multiple (check boxes)
- Pop-Up (a drop list).

When entering data, you can use the soft menu to apply one of the following input types to text boxes and fields (note, some elements do not use all input types). When a field is selected, the soft Menu label displays Qwerty. Open the menu and you see the input types numbered 1 through 5.

1. **Qwerty**. Any character on the keyboard
2. **123**. Any numeral
3. **ABC**. Any letter
4. **Predict**. Predicts next character based on prior input
5. **Symbols**. Opens a list of symbols; click to make a selection.
6. **Virtual**. Click keys on a virtual keyboard to enter data.

The Qwerty, 123, and ABC categories act as filters. For example, if you assign 123 to a field and you type "abc", nothing is entered in the field.

When you finish a demo, select the home button to return to the UIDemo launch page:

CustomItem. This demo features text fields, and text fields in table form. To type in the table, select a cell, then click to open a text entry panel and type your input. From the menu, select OK.

StringItem. Displays labels, a hyperlink, and a button. The soft menu action varies depending on the selected element.

Gauge. Interactive, non-interactive, indefinite and incremental gauges.

Alert. Uses pop-ups to display alerts. Set the alarm type and the length of the timeout from drop lists. Select the alert type and select the Show soft button.

ChoiceGroup. Radio buttons, check boxes, and pop-ups on one screen.

List. Select exclusive, implicit, or multiple to display the list type on a subsequent screen.

TextBox. Use text fields, radio buttons, check boxes, and pop-ups. Select a text box type and press the Show button.

TextField. Text fields with the six input types.

DateField. Set date and time using drop lists.

Ticker. A scrolling ticker.

3.7 Running IMP-NG Sample Projects

This section describes how to use demos created specifically for the IMP-NG platform (see Section 2.1.2, "IMP-NG"). Because IMP-NG is headless the only user interface is to observe application status in the emulator's external events generator, or in the Output window (or the console if you execute the demo from the command line).

With the exception of I2CDemo, the sample projects in this section can be run on the emulator or on a real device.

See the *Oracle® Java ME Embedded Getting Started Guide for the Reference Board Platform* and the *Oracle® Java ME Embedded Getting Started Guide for the Windows 32 Platform* These documents are available on the Java ME documentation site at `http://docs.oracle.com/javame/index.html`.

3.7.1 GPIODemo

This demo can be run on an emulator or a real device. The implementations are different, as the emulator uses the external events generator, and the real device supports direct interaction.

GPIODemo on the Emulator

- Run GPIO demo on an IMP-NG emulator.
- Click the GPIO tab. This view approximates the device actions.
- Choose Device > GPIO to open the external events generator, and click the GPIO tab. A single click on a button turns on LEDs indicating the button pushed and the pin affected. This information is also written to the Output window.

  Beneath each pin you can click the blue wave button to open the wave generator. The wave generator simulates the frequency and duration of the signal to the LED.
- Press Pin 5 (button 1) to turn on LED 1, press again to turn off LED 1.
- Press Pin 6 (button 2) to turn on LED 2, press again to turn off LED 2.
- Press Pin 7 (button 3) and check whether PORT 1's output value is 3. Press PIN 7 and check whether PORT 1's output value is 0.

GPIODemo on the Reference Board

The buttons are wired by GPIO. Assuming that you are looking at the board and the Reset button is on the bottom left:

- Reading from left to right, the LEDS are labelled: PH.3, PH.6, PH.7, PI.10, PG.6, PG.6, PG.8, and PH.2.
- The buttons, from left to right, are Reset, Wakeup, Tamper, and User.

  The Wakeup button corresponds to PH.3 (Listener 1 Pin 5).

  The Tamper button corresponds to PH.6 (Listener 2 Pin 6).

  The User button corresponds to PH.7 and PI.10 (Listener 3 Pin 7).
- Pressing Wakeup turns on the light, and releasing turns off light.
- When you press Tamper, the initial press does nothing (listener false). Releasing turns on light. After that, pressing turns light on and releasing turns light off. (Press listener is false and Release listener is true, as shown in the console output.)
- When you press User, the initial press does nothing. Release turns on two lights. After that, pressing turns two lights off and releasing turns two lights on. (Press listener is false and Release listener is true.)

3.7.2 I2CDemo

This demo is designed to work with the IMP-NG runtime for Windows 32. It has no user interaction.

- Launch the I2C demo.
- In the emulator, click the I2C tab.

  The demo acquires a slave named I2C_Echo, writes data to the slave, and retrieves it. The demo is successful if the Sent Data and Received Data matches.

3.7.3 NetworkDemoIMPNG

This demo is a headless version of Section 3.6.5.1, "Socket Demo."

This demo can be configured as a server or as a client by editing the application descriptor. You launch two instances of this demo, the first one acts as a server and the second one acts as a client. The client instance attempts to connect to the server instance and if the connection is successful they exchange a message.

NetworkDemoIMPNG on the Emulator

- Create two instance projects of the NetworkDemoIMPNG sample project.
- Right click on the first project and select Properties. In the Platform category choose the device IMPNGPhone1. In the Application Description category set the value of the property Oracle-Demo-Network-Mode to **Server** and click OK.
- Launch the first project. It opens on the emulator IMPNGPhone1 and waits for a connection.
- Right click on the second project and select Properties. In the Platform category choose the device IMPNGPhone2. In the Application Description category set the value of the property Oracle-Demo-Network-Mode to **Client** and click OK.
- Launch the second project. It opens on the emulator IMPNGPhone2.
- The client attempts to connect to the server. If successful, you see the following in the output tab of the first project (the server):

  ```
  Connection accepted
  Message received - Client messages
  ```

  The output of the second project (the client) shows the following:

  ```
  Connected to server localhost on port 5000
  Message received - Server String
  ```

NetworkDemoIMPNG on the Reference Board

You can run one of the instance projects on the board and the other in one of the emulators. Follow these steps to run the client on the board and the server in one of the emulators:

- Right click on the first project (the server) and select Properties. In the Platform category choose the device IMPNGPhone1 (the emulator) and click OK. In the Application Description category set the value of the property Oracle-Demo-Network-Mode to **Server** and click OK.
- Launch the first project (the server). It runs on the emulator and waits for a connection.
- Right click on the second project (the client) and select Properties. In the Platform category choose the device IMPNGExternalPhone1 (the board). In the Application Description category set the value of the property Oracle-Demo-Network-Mode to **Client** and click OK.

  In the Application Descriptor category set the value of the property Oracle-Demo-Network-Address to the IP address of the computer where NetBeans is running and click OK.
- Launch the second project (the client). It runs on the board and attempts to connect to the server. If successful, you see the following in the output tab of the first project (the server):

  ```
  Connection accepted
  ```

```
Message received - Client messages
```

The TCP log of the board (the client) shows the following:

```
Connected to server 10.0.0.10 on port 5000
Message received - Server String
```

3.7.4 PDAPDemoIMPNG

This demo is a headless version of the PDAPDemo file browser in Section 14.3.1, "Browsing Files."

PDAPDemoIMPNG on the Emulator

Follow these steps to run the demo on the emulator:

- Create test files and directories inside the emulator's file system:

  `Documents and Settings\`*user*`\javame-sdk\`*version*`\work\IMPNGPhone1\appdb\filesystem\root1`

- Right click on the project and select Properties. In the Platform category choose the device IMPNGPhone1 and click OK.
- Launch the project. It runs on IMPNGPhone1.
- On the emulator menu, select Device > File Connection to see a list of mounted file systems.
- Open a terminal emulator and create a raw connection to localhost on port 5001.
- A command line opens where you can browse the emulator's file system. You can use the following commands:
  - `cd` - change directory
  - `ls` - list information about the FILEs for the current directory
  - `new` - create new file or directory
  - `prop` - show properties of a file
  - `rm` - remove the file
  - `view` -View a file's content

  PDAPDemoIMPNG on the Board

  Follow these steps to run the demo on the board:

  - Right click on the project and select Properties. In the Platform category choose the device IMPNGExternalPhone1 and click OK.
  - Launch the project. It runs on the board.
  - Open a terminal emulator and create a raw connection to the IP address of the board on port 5001.
  - The command line that opens is the same as the one you use when you run the demo on the emulator.

  The file system of the demo is stored in the directory `java/`*IMletID* inside the SD card, where *IMletID* is a number that the AMS assigns to an IMlet during its installation.

4 Viewing and Editing Project Properties

All projects have properties. Some properties, such as the project's name and location cannot be changed, but other properties can be freely edited as work on your project progresses. To view or edit a project's properties, right-click the project node and select Properties. In the resulting window, you can view and customize the project properties. See the following topics:

- Section 4.1, "Project Properties"
- Section 4.2, "Platform Selection"
- Section 4.3, "Editing Application Descriptor Properties"
- Section 4.4, "Building a Project"
- Section 4.5, "Running Settings"

4.1 Project Properties

To view the project properties, right-click on a project, choose Properties, and select the Java ME category. The Java ME category enables you to select the emulator platform and configuration for the project, and it contains the following subcategories: library, obfuscation, packaging, preprocessor, preverification and signing.

4.2 Platform Selection

An emulator platform simulates the execution of an application on one or more target devices. To view this property page, right-click on a project and choose Properties and select the Java ME category.

Select a configuration from the table. You can edit the configuration and choose the SDK platform and the device that you want to use for this project.

For the emulator platform, be sure to select the 3.2 platform. You might have to use the dropdown menu to ensure the right version is selected.

By default, the devices in the device menu are also suitable for the platform type and emulator platform. The device you select is the default device for this project. It is used whenever you use the Run command. Your device selection influences the Device Configuration and Device Profile options, and the available optional packages.

Add the optional packages you want to include in this project. The selected APIs are automatically added to the project's classpath.

4.3 Editing Application Descriptor Properties

In Eclipse, the application descriptor is an element in the Package Explorer window. Double click on Application Descriptor in the Package Explorer under your project tree and edit the properties on the window that appears. There are different categories of properties on the tabs on the bottom: Overview, MIDlets, Optional, User Defined, Signing, etc.

4.4 Building a Project

When you build a project, the SDK compiles the source files and generates the packaged build output (a JAR file) for your project. You can build the main project and all of its required projects, or build any project individually.

In general you do not need to build the project or compile individual classes to run the project. By default, the SDK automatically compiles classes when you save them. You can use properties to modify the following build tasks:

- Section 4.4.1, "Compiling"
- Section 4.4.2, "Adding Libraries and Resources"
- Section 4.4.4, "Obfuscating"
- Section 4.4.3, "Creating JAR and JAD Files (Packaging)"
- Section 4.4.5, "Signing"

4.4.1 Compiling

To view this property page, right-click on a project and choose Properties. Choose the Java Compiler category. In this category and its subcategories you can choose the Java compiler options and other compilation options.

4.4.2 Adding Libraries and Resources

To view this property page, right-click on a project and choose Properties. In the Java ME category, choose the Library subcategory. Here you can add libraries to your project.

4.4.3 Creating JAR and JAD Files (Packaging)

To view this property page, right-click on a project and choose Properties. In the Java ME category, choose the Packaging subcategory.

4.4.4 Obfuscating

To view this property page, right-click on a project and choose Properties. In the Java ME category, choose the Obfuscation subcategory.

4.4.5 Signing

To view this property page, right-click on a project and choose Properties. In the Java ME category, choose the Signing subcategory.

4.5 Running Settings

To view this property page, right-click on a project and choose Properties. In the Properties window, choose Run/Debug Settings. The options shown depend on the platform. See Section 4.5.1, "MIDP Project Run Options."

4.5.1 MIDP Project Run Options

To set emulator command line options for a MIDP project, type in the command line switches. See Section 11.3, "Emulator Command Line Options."

To view the running settings page, right-click on a project and choose Properties. Select the Run/Debug Settings category and click on New. Select Java Application and click OK. In the window that appears you can adjust the run options for your project.

5 Working With Devices

The Oracle Java ME SDK emulator simulates a CLDC device on your desktop computer. The emulator does not represent a specific device, but it provides correct implementations of its supported APIs.

The Oracle Java ME SDK provides default device skins. A skin is a thin layer on top of the emulator implementation that defines the appearance, screen characteristics, and input controls.

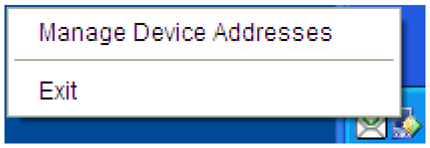
5.1 Emulating Devices

The SDK runs applications on an emulated device or a real device. Before you can run an application from the SDK, the Device Manager, which manages both emulated and real devices, must be running. When the Oracle Java ME SDK runs, the Device Manager automatically launches and starts detecting devices. The default devices shipped with the SDK are automatically found and displayed in the Device Selector window.

5.1.1 The Device Manager on Windows

The SDK uses the device manager to detect devices and displays the available devices in the Device Selector window. The Device Manager is a service and you can see it running in your Windows system tray. In the task manager, the process is labeled `device-manager.exe`.

You can right-click on the icon and select Exit to stop the service.



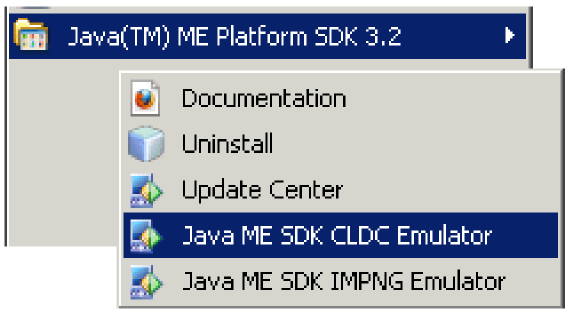
To restart the device manager, double-click *installdir*`\bin\device-manager.exe`. You can also start it from the command line as described in Section 11.1, "Run the Device Manager."

Choosing Manage Device Addresses opens the Device Address Manager. Enter an IP address and select Add to add a device. Select an address and click Remove if you have an address you no longer want to detect. The device will no longer be displayed in the Device Selector.

5.1.2 Starting an Emulator

Typically an emulator is launched when a Java ME SDK project is run from the IDE or the command line.

You can open an emulator without running an application from the IDE. From the Windows Start menu, select Programs > Java(TM) ME Platform SDK 3.2 and choose the desired emulator. You can also click the emulator shortcuts installed on your Windows desktop.


Java(TM) ME Platform SDK 3.2
Documentation
Uninstall
Update Center
Java ME SDK CLDC Emulator
Java ME SDK IMPNG Emulator

To run an application from the emulator, select Application > Run MIDlet Suite (or IMlet Suite). Provide the path to the application and any other information, and click OK.

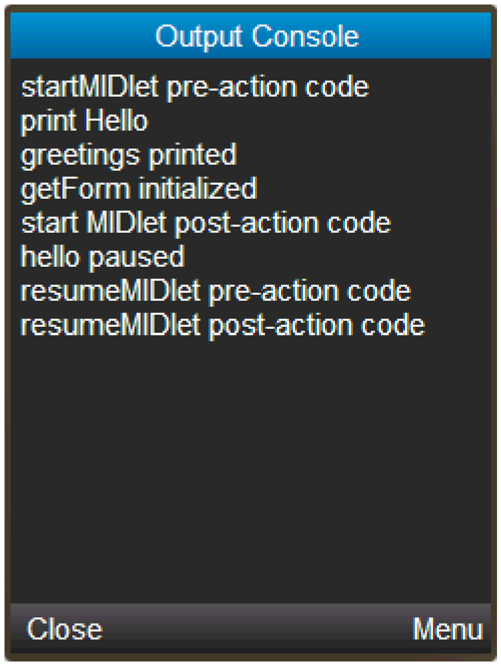
5.1.3 CLDC Application Management Software Home

The CLDC AMS home screen features three utilities:

- Install Application. This utility opens a form in which you can specify a URL (or a file path) for a JAD file to install.
- Manage Certificate Authorities. This feature displays the certificates authorities for the device. In this interface the white box indicates the certificate is checked (active). You can uncheck certificates that are not needed.
- Output Console. The output console displays system output statements from a running application. The application must write to the Java standard output console using, for example: `System.out.println("text");`

  Start the Output Console, then start your application. Use F7, Switch running MIDlet, to switch between the application and the Output Console.

  The Output Console is an application that consumes resources. See Section 5.5, "Changing the Maximum Number of Concurrent Applications."



```
startMIDlet pre-action code
print Hello
greetings printed
getForm initialized
start MIDlet post-action code
hello paused
resumeMIDlet pre-action code
resumeMIDlet post-action code
```

See Section 5.9, "Emulator Features" and Section 5.10, "Emulator Menus".

5.2 Adding a Real Device

The device selector can detect a device that has a compatible runtime. Typically this device has network capabilities and is connected to the computer running Java ME SDK.

1. To detect a physical device, click CTRL-A, or click the device icon at the top of the Device Selector window.
2. Type an IP address and click Next. Click Finish.

   You can also enter an IP address in the Device Manager, as described in Section 5.1.1, "The Device Manager on Windows".
3. The physical device is listed in the appropriate platform tree. By default the device has "ExternalPhone" appended to the name.

   For example, if an IMP-NG device is detected it is placed in the IMP-NG node and given the name IMPNGExternalPhone1.

   For an example of how to configure and work with a real device, see the *Oracle® Java ME Embedded Getting Started Guide for the Reference Board Platform*. This document is available on the Java ME documentation site: `http://docs.oracle.com/javame/index.html`.

5.3 Viewing Device Properties

The Device Selector window lists all available devices grouped by platform. If this window is not visible, select Window > Show View > Other > Java ME SDK Tools > Device Selector.

If no Java ME platform is registered in Eclipse, the Device Selector is empty. If you see this message at startup, it typically means device discovery is incomplete and you must wait a few seconds.

Each sub node represents a device skin. Two instances are provided for some CLDC devices, for example, JavaMEPhone1 and JavaMEPhone2. Instances of the same device have the same capabilities but unique names and phone numbers, making it easy for you to test communication between devices of the same type.

For Device names, see Section 6.4, "Oracle Java ME SDK Directories." The properties for each device skin are stored in XML files in your user work directory. See Table 6–1.

See also: Section 5.3.1, "Platform Properties," Section 5.3.2, "Device Information," and Section 5.3.3, "Device Properties."

5.3.1 Platform Properties

To view properties for Oracle Java ME SDK from the device selector, click on the top-level node. The Properties window is, by default, docked in the upper right portion of the user interface.

5.3.2 Device Information

In the Device Selector window, double-click a device node. The Device Information tab opens in the central Main window. It displays a picture of the device and displays details, supported hardware capabilities, keyboard support, supported media formats, and the supported runtimes.

5.3.3 Device Properties

In the Device Selector window, click a device node (such as JavaMEPhone1) to display the device properties. The Properties window is, by default, docked in the upper right portion of the user interface.

5.4 Setting Device Properties

In the Device Selector Window, click on a device and see the information in the Properties pane. Any properties shown in gray font cannot be changed. You can adjust the device properties shown in black. Only CLDC options can be adjusted.

5.4.1 General

This section lists general properties that can be changed.

Phone Number 1. You can set the phone number to any appropriate sequence, considering country codes, area codes, and so forth. If you reset this value, the setting applies to future instances. The number is a base value for the selected device.

Heapsize. The heap is the memory allocated on a device to store your application's objects. The Heapsize property is the maximum heap size for the emulator. You can choose a new maximum size from the drop-down menu.

Security Domain. Select a security setting from the drop-down menu.

JAM storage size in KB. The amount of space available for applications installed over the air.

Locale. Type in the locale as defined in the MIDP 2.0 specification: `http://jcp.org/en/jsr/detail?id=118`

Remove MIDlet Suite in execution mode. If this option is enabled, record stores and other resources created by the MIDlet are removed when you exit the MIDlet (assuming the MIDlet was started in execution mode).

5.4.2 Monitor

If enabled (checked), the Check boxes for Trace GC (garbage collection), Trace Class Loading, Trace Exceptions, and Trace Method Calls activate tracing for the device the next time the emulator is launched. The trace output is displayed at runtime in the user interface Output window. Trace Method Calls returns many messages, and emulator performance can be affected.

5.4.3 SATSA

See Section 17.1, "Card Slots in the Emulator."

5.4.4 Location Provider #1 and #2

These properties determine the selection of a location provider. Two providers are offered so that your application can test matching the location provider criteria.

If you select a property a short explanation is shown in the description area just below the Properties table. For more information on these values, see the Location API at `http://jcp.org/en/jsr/detail?id=179`.

5.4.5 Bluetooth and OBEX

Bluetooth. See Section 15.1, "Setting OBEX and Bluetooth Properties."

5.5 Changing the Maximum Number of Concurrent Applications

By default the CLDC runtime allows a maximum of five applications to run simultaneously. If you exceed the limit, you see the message, "No more concurrent applications allowed."

This number includes active applications, the Application Management Software (AMS), and any active on-device tooling agent (such as the profiler). You can close some applications, or increase the limit as follows:

1. In the Oracle Java ME SDK installation, locate the following file:

   installdir`\runtimes\cldc-hi\bin\jwc_properties.ini`

2. Locate the property MAX_ISOLATES. The value can be increased up to 15. However, you should be conservative as many applications running concurrently can affect performance.

5.6 Opening a Serial Port

In application code, you can use `Connector.open("comm:COM1")` to open a port on the device. On Windows, you can open a serial port such as COM1 or COM2.

5.7 Running a Project from the Device Selector

The SDK determines which open projects are suitable for a device. Right-click on the device and select a project from the context menu.

You can also launch the emulator to run a project from the command line, as explained in Section 11.3, "Emulator Command Line Options."

5.8 Running Projects Simultaneously on a Single Device

CLDC-based devices are capable of running multiple virtual machines. You can test this behavior in the emulator. Be sure the output window console is visible in the SDK. To test this feature, follow these steps:

1. Open the sample projects Games and AudioDemo.
2. In the device selector, choose a CLDC device and run Games. When the emulator launches run AudioDemo on the same device.

   As each MIDlet loads, the AMS automatically installs it.
3. In AudioDemo, launch the Audio Player, and play the JavaOne theme.

   Select AudioPlayer, then from the soft menu, select 1, Launch. Select JavaOne Theme and press the Play soft button.
4. In the emulator, choose Application > AMS Home, or press F4.

   Select Games. From the soft menu, select 1, Open. The music continues to play while you are able to simultaneously launch and play games.
5. Select Application > AMS Home, or press F4. Highlight AudioSamples, and from the soft menu, select 6, Bring to foreground. Press the Pause soft key. The music stops playing.
6. Select Application > AMS Home, or press F4. Highlight AudioSamples and from the soft menu, select 5, Open. Select Bouncing Ball from the list and press the Launch soft button. Select MIDI background and press the Play soft button.
7. Select Application > AMS Home, or press F4. Select Application > Switch Running MIDlet. Select Audio Player and press the Switch to soft button. You can press the Play soft button to resume the Audio Player.

5.9 Emulator Features

Figure 5–1, "Emulator Features" shows common emulator features available on emulators for the CLDC platform.

Device Name. Shown in the upper window frame. See Section 6–1, " Device Names."

Transmission Indicator. On the upper left of the emulator image, this blue light turns on when a transmission is occurring. Typically you see it when an application is installed over-the-air, or when a message is being sent or received. For example, when you receive a message from the WMA console.

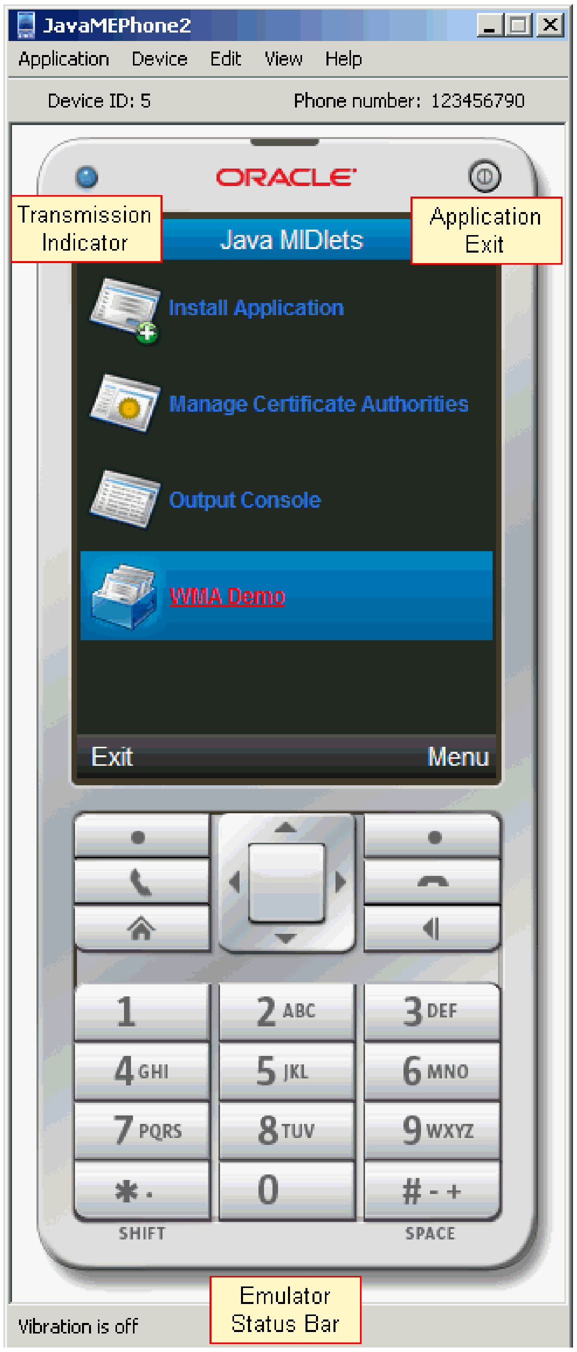
Menus. See Section 5.10, "Emulator Menus."

Device ID. See Table 6–1, " Device Names".

Exit Button. Pushing the button on the upper right of the emulator image has the same effect as selecting Application > Exit.

Emulator Status Bar. Information about the current system state is shown in the status bar at the bottom of the emulator window.

Figure 5–1 Emulator Features



5.10 Emulator Menus

The emulator for the CLDC platform has Application, Device, Edit, View, and Help menus.

5.10.1 Application

The Application menu is fully populated for the CLDC platform. The Application options are as follows:

Option	Accelerator	Description
Run MIDlet suite		Emulator interface for launching MIDlets.
AMS Home	F4	Exit the current application and return to the Application Management Software home.
Stop	F10	Stops the currently running MIDlet.
Change Locale		This option only works with localized MIDlets. Enter a locale identifier. The format is similar to Java SE 6, as follows: 2-letter-lang-code *separator* 2-letter-country-code For example, en-US, cs-CZ, zh-CN, ja-JP. The separator can be a dash or an underscore.
Resume	F6	Resume a suspended application.
Suspend	F5	Pause a running application. **Do not use this option if you are running the memory monitor.**
Switch Running MIDlet	F7	When you have multiple MIDlets running, toggle between them. You see a list of running MIDlets and you can chose the one you want to switch to. See Section 5.8, "Running Projects Simultaneously on a Single Device."
Exit	Exit button on emulator upper right	Close the emulator process and stop the build process (or processes).

5.10.2 Device

This menu is available on CLDC platforms only.

5.10.2.1 Messages

Choose Device > Messages to see what is written in the message area. This is the emulator's Inbox. The Inbox displays WMA messages that are addressed to the device, not an application on the device. Messages are sent to this interface in the following cases:

- an MMS message is sent without an AppID in the address
- an SMS message is sent without a port in the address (or the port number is 0)
- an SMS text message is sent with a port in the address, but there is not a Java ME application listening on the specified port

To test sending messages to the inbox use the WMA Console in Eclipse, or from the command line, use `wma-tool.exe` to send SMS messages. Note, `wma-tool.exe` requires an AppID for MMS, so `wma-tool` can not be used to send an MMS.

5.10.2.2 Landmark Stores

Choose Device > Landmark Stores to open the Landmark Store utility. In this interface you can view and edit a landmark store installed as part of an application. You can also create a new landmark store, define landmarks, define landmark categories, and assign landmarks to categories.

5.10.2.3 Orientation

This option is only visible for the CLDC emulator.

Use this feature to test your application's ability to display in portrait and landscape formats. The default is 0 degrees. Change the orientation to 90, 180, or 270 degrees. You can also rotate 90 degrees clockwise (F8) or counterclockwise (F9) from the current position.

5.10.2.4 External Events Generator

The External Events Generator provides a way to interact with an application by injecting events. The interaction may be through a user interface, or through a script file. The following menu options each have a tab on the External Events Generator. The use of the External Events Generator is addressed in the discussion for each JSR.

- **Contactless Communication**. This option is only visible for the CLDC emulator. See Section 25.1, "Using ContactlessDemo."
- **File Connection**. Section 14.1, "FileConnection API."
- **GPIO**. This General Purpose Input Output (GPIO) option is only visible for the IMP-NG emulator.

  By default this tab displays ports and pins for a specific device described in Section 3.7.1, "GPIODemo." You can create a custom skin to represent a different device. Section 5.11.3.1, "General Purpose Input Output (GPIO)."

  See the Device Access API (*installdir*`\docs\api\deviceaccess`) for a description of the GPIO interface.
- **Location**. Section 18.1, "Setting the Emulator's Location at Runtime."
- **MMIO**. The memory-mapped I/O (MMIO) option is only visible for the IMP-NG emulator. From the Device dropdown list, choose one of the default devices:
  - **TEST_DEVICE**. A little Endian device that contains all block types: byte, short, int, long, and block. Writes to this device also affect BIG_ENDIAN_DEVICE which shares the address space.
  - **BIG_ENDIAN_DEVICE.** A Big Endian device that shares the address space with the TEST_DEVICE, therefore it contains the same memory blocks. Writes to this device also affect TEST_DEVICE which shares the address space.
  - **WDOGLOG**. A Little Endian device that supports only the block type.
  - **RTC**. STM32F2xx RTC device. A Little Endian device that supports only the int type.

  If you are using a custom skin created with the Skin Creator (see Section 5.11, "Using the Custom Device Skin Creator") the device list might include additional devices.

  See the Device Access API (*installdir*`\docs\api\deviceaccess)`and the Embedded Support API (*installdir*`\docs\api\embedded-support-api)` for descriptions of the MMIO interface.
- **Sensors**. This option is only visible for the CLDC emulator. See Section 24.2, "Using a Mobile Sensor Project" and Section 24.3, "Creating a Sensor Script File."
- **SPI**. In the default SPI implementation buffered written data can be read from the Slave named SPI. If you have created a custom implementation with the Skin Creator, the Slave dropdown list might have additional slaves

5.10.3 Edit

The Edit menu provides basic editing utilities for the CLDC platform.

Option	Accelerator	Description
Copy	CTRL-C	Copy selected material to the paste buffer.
Cut	CTRL-X	Move selected material to the paste buffer.
Paste	CTRL-V	Insert the contents of the paste buffer.

5.10.4 View

The only View option available is Always On Top.

Option	Description
Always On Top	Keeps the emulator in the foreground. This is especially useful when you are running multiple emulator instances and you want to see them all and send messages between devices.

5.10.5 Help

The Help menu displays an abbreviated helpset specifically for the emulator window.

5.11 Using the Custom Device Skin Creator

With the Custom Device Skin Creator you can create your own skins. The appearance of the custom skins is generic, but the functionality can be tailored to your own specifications.

5.11.1 Creating a New Custom Device Skin

Follow these steps to create a new custom device skin.

1. Launch the skin creator from the your installation’s bin directory. For example:

   ```
   C:\Java_ME_platform_SDK_3.2\bin\skin-creator.exe
   ```

   The custom device tree displays Java ME platforms and custom devices, if any.
2. Select a platform and click the New... button.
3. Specify a Name and Description.
4. Change the default configuration to match your specifications. If you are creating an IMP-NG skin, see Section 5.11.3, "IMP-NG Skin Options" for description of the IMP-NG tabs.
5. Click OK. Your skin is added to the custom device tree.

   Your custom skin definition is saved in *installdir*`\toolkit-lib\devices`. It can be used to run projects from the IDE or from the command line.

   The custom device tree affects what appears in the Device Selector. For example, if you don’t want a custom skin to appear in the device selector, you must remove it from the Skin Creator’s custom device tree.
6. To detect the device, select Window > Preferences >Java ME > Device Management.

   Click Manual Install. The Manual Device Installation window opens.

   In the "Specify search directory" field, browse to the your Java ME SDK installation directory. For example: `C:\Java_ME_platform_SDK_3.2`. When the devices are detected, click Finish to close the window, then click OK.

7. The detected device(s) appear in the Device Selector.

5.11.2 Managing Custom Skins

Custom skins should always be managed using the Custom Device Skin Creator. Using the tool ensures that your skin will be properly detected and integrated with the Oracle Java ME SDK.

- **Edit**. Select a device to change, and click Edit.
- **Clone**. Select a device to copy, and click Clone. To prevent confusion, be sure to provide a unique name.
- **Export**. Select a device to save, and click Export.

  When a custom device is created it is saved in *installdir*`\toolkit-lib\devices`, therefore you could lose your device if you reinstall.

  An exported device is stored in a .zip file and saved in the user's My Documents directory (typically *userhome*`\My Documents`).
- **Import**. Select a node in the custom device tree and click Import. Choose a .zip file created with the Export command.
- **Remove**. Select a device to delete and click Remove. This action removes the skin from the device selector and deletes it from its default location on disk (if you have exported the skin to a different location you can, import it at a later time).

5.11.3 IMP-NG Skin Options

When you create a new IMP-NG skin you can use the default implementation or create your own custom implementation for the interfaces discussed in this section.

See the following resources for in-depth descriptions of the IMP-NG interfaces or devices:

- *Device Access API (installdir*`\docs\api\deviceaccess)`
- *Embedded Support API (installdir*`\docs\api\embedded-support-api)`
- For more information about default devices look at an IMP-NG property file:

  userhome`\javame-sdk\`*version*`\work\IMPNGDevice1\device.properties`

5.11.3.1 General Purpose Input Output (GPIO)

A GPIO port is a platform-defined grouping of GPIO pins that may be configured for output or input. Output ports are both writable and readable while input ports are only readable. Note that GPIO pins that are part of a GPIO port cannot be retrieved nor controlled individually as GPIOPin instances.

Click Add Port to add entries to the Ports table. Each item in a Ports table row is editable. Check Output to set the port direction to output.

A GPIO pin may be configured for output or input. Output pins are both writable and readable while input pins are only readable. Note, an input listener can only be assigned to a pin set for input.

Click Add Pin to add a row to the Pins table. The ID and name are editable. Click Output to make a pin both readable and writable. Click in the Initial Value column to toggle the Initial Value from Off to On.

5.11.3.2 Inter-Integrated Circuit (I2C) and Serial Peripheral Interface (SPI)

The process for configuring I2C and SPI interfaces is very similar.

On an I2C bus, data is transferred between the I2C master device and an I2C slave device through single or combined messages. On an SPI bus, data is transferred between the SPI master device and an SPI slave device in full duplex. That is, data is transmitted by the SPI master to the SPI slave at the same time data is received from the SPI slave by the SPI master. See the *Embedded Support API* for more details.

- Select Sample Echo to choose the default bus implementation. This default implementation simply reads buffered written data from the slave.
- Choose Custom to specify your own bus implementation.

  1. Supply your bus implementation JAR file and the name of the Java class that implements the bus.

     For I2c the bus is:

     `com.oracle.jme.toolkit.deviceaccess.i2c.I2CSlaveBus`

     For SPI the bus is:

     `com.oracle.jme.toolkit.deviceaccess.spi.SPISlaveBus`

  2. To add Slaves, click Add and specify an ID and Name. For SPI specify the Word Length as well.

5.11.3.3 Memory-Mapped I/O (MMIO)

The default devices are described in Section 5.10.2.4, "External Events Generator."

If you want to provide your own MMIO emulation, you must specify a custom handler.

Supply your implementation JAR file and the name of the Java class that implements `com.oracle.jme.toolkit.deviceaccess.mmio.MMIOHandler`. For comparison, the default JAR file is:

installdir`\toolkit-lib\devices\IMPNGDevice\code\emulator_deviceaccess_mmio-sample-handler.jar`

To add devices to the custom MMIO implementation, use the Devices and Device Memory tables as follows:

1. Click Add Device to add a row to the Devices table.
   - A default ID is assigned but you can double-click in the ID column to edit the value.
   - A default Name is supplied, but it can also be edited.
   - In the Byte Ordering column, make a selection from the dropdown list.
2. Click a row in the Device table to select a Device.
3. Click Add Memory.
   - In the Type column, make a selection from the dropdown list. Double-click to edit the Address column entries.

     If the type is Block you can double-click to edit the Size column entries as well as the Address column entries.
4. Click OK.

6 Finding Files in the Multiple User Environment

The Oracle Java ME SDK can be installed on a system running a supported operating system version. All users with an account on the host machine can access the SDK. This feature is called the Multiple User Environment.

Note: The Multiple User Environment supports access from several accounts. It does not support multiple users accessing the SDK simultaneously. See Section 6.1, "Switching Users."

To support multiple users the SDK creates an installation directory that is used as a source for copying. This document uses the variable *work* to represent the SDK working directory and *installdir* to represent the Oracle Java ME SDK installation directory. Each user's personal files are maintained in a separate working directory named `javame-sdk` that has a subdirectory for each version installed.

- Section 6.2, "Installation Directories"
- Section 6.3, "Eclipse User Directories"

To locate logs, see Section 12.1, "Device Manager Logs" and Section 12.2, "Device Instance Logs."

6.1 Switching Users

Multiple users cannot run the SDK simultaneously, but, you can run the SDK from different user accounts on the SDK host machine. When you switch users, you must close the SDK and exit the Device Manager, as described in Section 5.1.1, "The Device Manager on Windows." A different user can then launch the SDK and own all processes.

6.2 Installation Directories

The SDK directory structure conforms to the Unified Emulator Interface Specification (`http://www.oracle.com/technetwork/java/javame/documentation/ueispecs-187994.pdf`), version 1.0.2. This structure is recognized by all IDEs and other tools that work with the UEI.

The installation directory has the following structure:

- `bin`. The bin directory contains the following command line tools. The default location of the bin directory is:

installdir\bin

- `cref`.Java Card simulator for working with SATSA JSR 177. See Section 17.2, "Java Card Platform Simulator (cref)."
- `device-address` is a tool for viewing, adding, and removing devices that the SDK is not able to discover automatically. See Section 11.2, "Manage Device Addresses (device-address)."
- `device-manager`. The device manager is a component that must be running when you work with Oracle Java ME SDK. After installation it starts as a service, and it automatically restarts every time your computer restarts. See Section 5.1, "Emulating Devices.".
- `emulator`. UEI compliant emulator. See Section 11.3, "Emulator Command Line Options."
- `jadtool`. Tool for signing MIDlets. See Section 11.6.2, "Sign MIDlet Suites (jadtool)."
- `mekeytool`. Management of ME keystores. See Section 11.6.3, "Manage Certificates (MEKeyTool)."
- `preverify`. The Java ME preverifier.
- `wma-tool`. A command line tool for sending and receiving SMS, CBS, and MMS messages. See Section 19.3, "Running WMA Tool."
- `wscompile`. Compiles of stubs and skeletons for JSR 172. See Section 11.7, "Generate Stubs (wscompile)."

■ `docs`. Release documentation.

■ `legal`. License and copyright files.

■ `lib`. JSR JAR files for compilation.

■ `runtimes`. CLDC and IMP-NG runtime files.

■ `toolkit-lib`. Java ME SDK files for configuration and definition of devices and UI elements. Executables and configuration files for the device manager and other SDK services and utilities.

6.3 Eclipse User Directories

By default, the Eclipse default workspace file is located at:

`C:\Documents and Settings\`*userhome*`\workspace`

You can use the File > Switch Workspace command to change the location.

6.4 Oracle Java ME SDK Directories

This documentation sometimes uses *userhome* to represent the root location of user files.

■ The `javame-sdk` directory contains device instances and session information. If you delete this directory, it is re-created automatically when the device manager is restarted.

userhome`\javame-sdk\3.2`

■ Device working directories

userhome`\javame-sdk\3.2\work\`*devicename*

The named subdirectories each correspond to an emulation device, as described in Table 6–1. Any detected real devices are also added to this directory space. Device detection is described in Section 5.2, "Adding a Real Device."

Table 6–1 Device Names

Device	Platform	Emulator #
ClamshellJavaMEPhone1	CLDC	0
IMPNGPhone1	CLDC	2
IMPNGPhone2	CLDC	3
JavaMEPhone1	CLDC	4
JavaMEPhone2	CLDC	5

7 Profiling Applications

The Oracle Java ME SDK supports performance profiling for Java ME applications. The profiler keeps track of every method in your application. For a particular emulation session, it figures out how much time was spent in each method and how many times each method was called.

The SDK supports offline profiling. Data is collected during the emulation session. As you view the snapshot you can investigate particular methods or classes.

Note: This feature might slow the execution of your application.

7.1 Collecting and Saving Profiler Data in the IDE

This procedure describes interactive profiling. To run profile an application from the command line, see Section 11.3.3, "Command Line Profiling.".

Note: The profiler maintains a large amount of data, so profiled MIDlets place greater demands on the heap. To increase the Heapsize property, see Section 5.4, "Setting Device Properties."

1. Select a project.
2. Select menu Run > Profile or right click project Profile As > CPU Profiler.
3. Interact with the application MIDlet(s) as you normally would.
4. Exit the MIDlet.

   You are asked if you would like to transfer the profiling data to a profiler. Choose Yes.

   The profile data is automatically displayed in a tab labeled cpu:*time*, where *time* is the time the data was displayed.

   In the IMP-NG emulator the data transfer occurs when you press the Stop button.
5. To export the profile data, choose File > Save As. The file is saved as a PRF file in the project directory in your workspace.

7.2 Loading a .prf File

A previously exported `.prf` file (Section 7.1, "Collecting and Saving Profiler Data in the IDE") can be loaded at a later time.

Follow these steps to retrieve profile data:

1. Select File > Open File...
2. Browse to the `.prf` file.

The Profiler opens in its own tab labeled cpu:*filename*.

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

8 Network Monitoring

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.

Note: Network monitoring supports only the HTTP and HTTPS protocols.

Networking monitoring works for emulators only (it is not supported for real devices).

8.1 Monitor Network Traffic

Follow these steps to activate the network activity for an application.

1. Run > Network Monitor or right click Project, select Profile As > Network Monitor
2. Start your application.

   When the application makes any type of network connection, information about the connection is captured and displayed in the Network Monitor tab.

   The top frame displays a list of messages. Click a message to display its details in the bottom frame.

   In the Hex View, message bodies are shown as raw hexadecimal values with the equivalent text. To avoid memory issues, the Hex view is currently limited to 16kB of data.

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.

8.2 Filter or Sort Messages

Filters are useful for examining some subset of the total network traffic.

- In the [Select Devices] list check only the devices you want to view.
- In the [Select Protocols] list check only the protocols you want to view. The protocols listed reflect what is currently installed on the device.

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

To arrange the message tree in a particular order, click on the Sort By combo box and choose a criteria.

Note: Sorting parameters are dependent on the message protocol you choose. For example, sorting by time is not relevant for socket messages.

8.3 Save and Load Network Monitor Information

Use File > Save As to save your network monitor session. The session is saved as a Network Monitor Snapshot file and is added to the project tree, just like the profiling information.

To load a network monitor session, choose File > Open File... and browse to the `.nmd` file you saved.

Note: To avoid memory issues, the Hex view display is currently limited to 16kB of data.

9 Lightweight UI Toolkit

The Lightweight UI Toolkit (LWUIT) is a lightweight widget library inspired by Swing but designed for constrained devices such as mobile phones and set-top boxes. Lightweight UI Toolkit supports pluggable theme-ability, a component and container hierarchy, and abstraction of the underlying GUI toolkit. The term lightweight indicates that the widgets in the library draw their state in Java source without native peer rendering.

9.1 LWUIT and the Java ME SDK

LWUIT is an open source project whose source is available at `http://lwuit.java.net`.

Java ME SDK 3.2 ships with the LWUIT 1.5 library which is installed as plugin. For information on this release, see the product page at:

`http://www.oracle.com/technetwork/java/javame/javamobile/download/lwuit/index.html`

The *Lightweight UI Toolkit Developer's Guide* is available in PDF and HTML formats:

PDF: `http://download.oracle.com/javame/dev-tools/lwuit-1.5/LWUIT_Developer_Guide.pdf`

HTML:
`http://download.oracle.com/javame/dev-tools/lwuit-1.5/devguide/toc.htm`

As an open source project, LWUIT has an independent release schedule. The Java ME SDK Update Center updates LWUIT when an official binary is released.

It is possible that you might want to use a development version of the LWUIT library. You can add a newer version as described in Section 9.3, "Add a Different LWUIT Library."

9.2 LWUIT Resource Editor

The Resource Editor is an independent GUI tool for opening, creating, and editing resource packages for LWUIT. To obtain the Resource Editor, go to `http://www.oracle.com/technetwork/java/javame/javamobile/download/lwuit/index.html`, download the distribution, and use the tool that is included there.

The Resource Editor has its own help, and tutorials that are accessed from the Resource Editor's Help menu. These articles link back to the LWUIT blog. For traditional documentation, see the "Resources" chapter in the Developer's Guide mentioned in Section 9.1, "LWUIT and the Java ME SDK."

9.3 Add a Different LWUIT Library

To add the LWUIT library to an Eclipse project, follow these steps:

1. Choose Project > Properties to open the Project Properties dialog.
2. Select the Java Build Path category.
3. Select the Libraries tab.
4. Press Add External JARs then browse to the appropriate LWUIT JAR file for the platform.

9.4 LWUIT Demos

This release provides new and updated demos and sample code. Most of these demos are self-evident user interface samples.

Note: Many LWUIT demos access common internet sites and services through publicly available APIs. To see the demos working as intended you might have to change your proxy settings or create an exception in your antivirus software.

- **LWUITBrowser**

  From the menu, select Help for an explanation of this demo.

- LWUITDemo

  This application has demos for many features. From the Menu choose About for a description of the demo. Choose a demo and press the Help soft button for an explanation.

- LWUITIODemo

  This application implements IO features. For example, type LWUIT in the Search box, choose blog from the Type menu, and press Go. Click the search results to load the page into your system's default browser.

- LWUITMakeover

  This demo features a search performed by distance, title, rating, or relevance. Search results can be mapped. To "makeover" the demo by choose a different theme from the Menu.

- LWUITSpeed

  This demo tests drawing speed for different components. Press the Start button to cycle through a series of animations. To change the performance you can edit the frame rate in SpeedMIDlet.java. You can also affect the performance by changing the emulator's heap size. In the Device Selector, right-click on the device, select Properties from the context menu, and change the Heapsize value.

- LWUITTimeZone

  This application shows a contacts list and provides date and time information for contacts displayed on the home page. Use + to add contacts and - to remove them. Press the sun symbol to toggle the time format between 24 hour time and civilian time.

- LWUITTipster

The demo is a simple tip calculator. The default service is restaurant staff. To change the service type, click the up arrow to highlight the service types. Use the right or left arrows to highlight a service type, then click the select button.

10 Security and MIDlet Signing

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

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.

The general process to create a cryptographically signed MIDlet suite is as follows:

1. The MIDlet author, probably a software company, buys a signing key pair from a certificate authority (the CA).
2. The author signs the MIDlet suite with the signing key pair and distributes their certificate with the MIDlet suite.
3. When the MIDlet suite is installed on the emulator or on a device, the implementation verifies the author's certificate using its own copy of the CA's root certificate. Then it uses the author's certificate to verify the signature on the MIDlet suite.
4. After verification, the device or emulator installs the MIDlet suite into the security domain that is associated with the CA's root certificate.

For definitive information, consult the MIDP 2.0 specification. For an overview of MIDlet signing using the Oracle Java ME SDK, read the article Understanding MIDP 2.0's Security Architecture.

If you need more background on public key cryptography, see `MIDP Application Security 1: Design Concerns and Cryptography` and the `Java Cryptography Architecture Specification`.

See the following topics:

- Section 10.1, "Security Domains"
- Section 10.4, "Managing Keystores and Key Pairs"
- Section 10.3, "Signing a Project With a Key Pair"

10.1 Security Domains

The SDK supports the following security domains:

`minimum`. All permissions are denied to MIDlets in this domain.

`maximum`. All permissions are granted to MIDlets in this domain. Maximum is the default setting.

`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.

`operator`. All permissions are denied to MIDlets in this domain.

`manufacturer`. Intended for MIDlet suites whose credentials originate from the manufacturer's root certificate.

10.2 Setting Security Domains

In the SDK, when you use Run Project via OTA your packaged MIDlet suite is installed directly into the emulator where it is placed in a security domain. The emulator uses public key cryptography to determine the appropriate security domain.

- If the MIDlet or MIDlet suite is not signed, it is placed in the default security domain.
- If the MIDlet or MIDlet suite is signed, it is placed in the protection domain that is associated with the root certificate of the signing key's certificate chain. See Section 10.3, "Signing a Project With a Key Pair."

  If your project is a MIDlet suite, the entire suite is signed (the individual MIDlets contained within are not).

10.2.1 Specify the Security Domain for an Emulator

Follow these steps to specify the security domain for an emulated device.

1. Select the device in the device selector.
2. In the Properties Window, expand the General properties, and for the Security Domain option, choose a domain from the dropdown list.

10.2.2 Specify the Security Domain for a Project

Follow these steps to set a MIDlet Suite’s security domain at runtime.

1. Right-click on the package and select Run As > Run Configurations... from the context menu.
2. Choose the project’s JAD file, then select the Emulation tab.
3. Specify the device and the security domain, and click Run.

You can also sign your MIDlet or IMlet with JADtool (Section 11.6.2, "Sign MIDlet Suites (jadtool)".

10.3 Signing a Project With a Key Pair

Devices use signing information to check an application's source and validity before allowing it to access protected APIs. For test purposes, you can create a signing key pair to sign an application. A key pair consists of the following:

- A private key that is used to create a digital signature, or certificate.
- A public key that anyone can use to verify the authenticity of the digital signature.

You can create a key pair as described in Section 10.4, "Managing Keystores and Key Pairs."

Follow these steps to sign a Java ME package in Eclipse.

1. In the Package view right-click on a package and select Properties to open the Properties dialog.
2. In the Java ME category, select Signing. For help with this page, view the following help topic: Java ME Development User Guide > Reference > Property Pages > Java ME.
3. Click Enable project specific settings. Specify a keystore and a password option.
4. Click OK.

To obfuscate code, see Section 4.4.4, "Obfuscating."

10.4 Managing Keystores and Key Pairs

The Oracle Java ME SDK command line tools described in Section 11.6.3, "Manage Certificates (MEKeyTool)" manage an emulator's list of root certificates.

Oracle Java ME SDK ships a default keystore named `_main.ks` in *installdir*`\runtimes\cldc-hi\appdb`. This keystore is automatically copied from your installation's default location to each instance of the default devices (the emulators). These instances are typically stored in:

`C:\Documents and Settings\`*username*`\javame-sdk\3.2\work\`*devicename*

Real devices have similar lists of root certificates, although you typically cannot modify them. When you deploy your application on a real device, you must use signing keys issued by a certificate authority whose root certificate is present on the device. This makes it possible for the device to verify your application.

In Eclipse you can also use MTJ utilities to manage keystores as described in Section 10.3, "Signing a Project With a Key Pair." You can also use the `-import` option to import certificates from these keystores as described in Section 11.6.3, "Manage Certificates (MEKeyTool)."

10.5 Command Line Samples

This section is a summary of command line samples for keystore and certificate tasks. For the full syntax of keytool, see:
`http://docs.oracle.com/javase/7/docs/technotes/tools/solaris/keytool.html`

These samples show literal paths on a sample system. You can replace the paths and options as you see fit. These samples contain linefeeds to accommodate the book format. In practice they commands should be entered on a single line.

Generate a keypair.

As mentioned in Section 10.4, "Managing Keystores and Key Pairs," Java ME includes a default keystore used for the emulators. For test purposes you can also make your own keypair containing a new keystore and a certificate. For example:

```
"C:\Program Files\Java\jdk1.6.0_35\bin\keytool" -genkeypair -alias mykp
-keyalg RSA -keysize 1024 -validity 365 -dname "cn=dlp, L=Santa Clara, S=CA"
-keypass 123456 -keystore D:/temp/mykeystore.ks -storepass 654321
```

List the keypair.

```
C:\Progra~1\Java\jdk1.6.0_35\bin\keytool -list -alias mykp
-keystore mykeystore.ks -storepass 654321
```

Export a certificate to a file.

```
C:\Progra~1\Java\jdk1.6.0_35\bin\keytool -exportcert -alias mykp
-keystore mykeystore.ks -storepass 654321 -file d:\temp\mykpcert
```

Print the certificate file.

```
C:\Progra~1\Java\jdk1.6.0_35\bin\keytool -printcert -file d:\temp\mykpcert
```

Import the keystore (including your certificate) into the default keystore.

For a description of MEKeyTool, see Section 11.6.3, "Manage Certificates (MEKeyTool)." In this example `mekeytool` is launched from the Java ME SDK installation `bin` directory.

```
mekeytool -import -alias mykp -keystore D:\temp\mykeystore.ks -MEkeystore
D:\temp\_main.ks -storepass 654321
```

11 Command Line Reference

This topic describes how to operate the Oracle Java ME SDK from the command line and details the command line tools required to build and run an application.

- Section 11.1, "Run the Device Manager"
- Section 11.2, "Manage Device Addresses (device-address)"
- Section 11.3, "Emulator Command Line Options"
- Section 11.4, "Build a Project from the Command Line"
- Section 11.5, "Packaging a MIDlet Suite (JAR and JAD)"
- Section 11.6, "Command Line Security Features"
- Section 11.7, "Generate Stubs (wscompile)"

11.1 Run the Device Manager

The device manager is a component that runs as a service. It detects devices (real or emulated) that conform to the Unified Emulator Interface Specification (`http://www.oracle.com/technetwork/java/javame/documentation/ueispecs-187994.pdf`), version 1.0.2. The Device Manager automatically restarts every time you use the SDK. You can manually launch the device manager from a script or a command line.

installdir`\bin\device-manager.exe`

To see a log of activities, launch the device manager with the `-XenableOutput` option.

11.2 Manage Device Addresses (device-address)

installdir`\bin\device-address` is a tool for viewing, adding, and removing devices that the SDK is not able to discover automatically. The Microsoft device emulator is an example of such a device. The syntax is:

Table 11–1 Device Address Commands

Command	Action
`add` *address_type address*	Add the specified address.
`del` *address_type address*	Delete the specified address.
`list`	List all addresses.
`list` address_type	List the specified address type.

For example, the following command adds a device:

```
installdir\bin\device-address.exe add ip 192.168.1.2
```

11.3 Emulator Command Line Options

You can launch the emulator independent of the GUI using `bin\emulator`. The syntax is as follows:

```
emulator options
```

The general options are as follows:

Table 11–2 Emulator Commands

Command	Action
`-classpath` *path* `-cp` *path*	Specifies a search path for application classes. The path consists of directories, ZIP files, and JAR files separated by semicolons.
-D*property=value*	Sets a system property value.
`-help`	Display a list of valid options.
`-version`	Display version information about the emulator.
`-Xdevice:`*devicename*	Run an application on the emulator using the given device instance name.
`-Xquery`	Print emulator skin information on the standard output stream and exit immediately. The information includes the skin name, screen size, and other capabilities.

This is a simple example of running the emulator from the command line:

```
emulator.exe -Xdescriptor:"C:\Program Files\Java_ME_platform_SDK_3.2\apps\Games\dist\Games.jad"
-Xdevice:JavaMEPhone2
```

`emulator.exe` also supports Section 11.3.1, "MIDlet Options" and Section 11.3.2, "Debugging and Tracing Options."

11.3.1 MIDlet Options

Options for running MIDlets in the emulator are as follows:

- `-Xautotest:`*JAD-file-URL*

  Run in autotest mode. This option installs a MIDlet suite from a URL, runs it, removes it, and repeats the process. The purpose is to run test compatibility kits (TCKs) with the emulator, using a test harness such as JT Harness (`http://jtharness.java.net`), or Java Device Test Suite (JDTS `http://www.oracle.com/technetwork/java/javame/javadevice-140362.html`). For example:

  ```
  emulator -Xautotest:http://localhost:8080/test/getNextApp.jad
  ```

  Given the above command, `-Xautotest` causes the emulator to repeatedly install, run, and remove the first MIDlet from the MIDlet suite provided through the HTTP URL. When the emulator starts, it queries the test harness, which then downloads and installs the TCK MIDletAgent.

- `-Xdescriptor:`*jad-file*

  Install a MIDlet, run it, and uninstall it after it finishes.

- `-Xdomain:`*domain-name*

  Set the MIDlet suite's security domain.

The `Xjam` argument runs an application remotely using the Application Management Software (AMS) to run over-the-air (OTA) provisioning. If no application is specified with the argument, the graphical AMS is run.

- `-Xjam[:=<JAD-file-url> |force|list|storageNames| run=[<storageNames>|<StorageNumber>]|remove=[<storage name>|<storage number> | all]]`

  Installs the application with the specified JAD file onto a device.

  - `force`. If an existing application has the same storage name as the application to be installed, `force` removes the existing application before installing the new application.
  - `list`. List all the applications installed on the device and exit. The list is written to standard output before the emulator exits.
  - `storageNames`. List all applications installed on the device. The list is written to standard output before the emulator exits. Each line contains one storage name in numerical order. The list contains only the name so the order is important. For example the first storage name must be storage number 1.

- `-Xjam:run=[`<*storage-name*> | <*storage-number*>`]`

  Run a previously installed application. The application is specified by its valid storage name or storage number.

- `-Xjam:remove=[`<*storage-name*> | <*storage-number*> | `all]`

  Remove a previously installed application. The application is identified by its valid storage name or storage number. If `all` is supplied, all previously installed applications are removed.

- `transient=`*jad-file-url*

  If specified, `transient` is an alias for installing, running, and removing the application with the specified JAD file.

This example illustrates OTA installation:

```
emulator -Xjam:install=http://www.myserver.com/apps/MyApp.jad
         -Xdevice:JavaMEPhone2
```

The above command returns the ID of the installed application. When you obtain the ID you can run it with: `emulator=Xjam:run=`*ID*

See also Section 11.3, "Emulator Command Line Options" and Section 11.3.2, "Debugging and Tracing Options."

11.3.2 Debugging and Tracing Options

You can use the following options with the emulator for debugging and tracing CLDC projects.

- `-Xdebug`

  Enable runtime debugging. The `-Xrunjdwp` option must be called to support `-Xdebug`.

- `-Xrunjdwp:`*debug-settings*

Start a Java debug wire protocol session, as specified by a list of comma-separated debug settings. Both `-Xrunjdwp` and `-Xdebug` must be called.

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. 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).
- `suspend=`*{y|n}* **-** The possible values are `y` and `n`.

  When `suspend` is set to *n*, the application starts immediately and the debugger can be attached at any time during its run.

  When suspend is set to *y*, the application does not start until a debugger attaches to the debugging port and sends a resume command, so an application can be debugged from the very beginning.

This example shows debugging:

```
emulator.exe -Xdevice:JavaMEPhone1 -Xdebug -Xrunjdwp:transport=dt_socket,suspend=n,
server=y,address=51307 -Xdescriptor:..\apps\Games\dist\Games.jad -Xdomain:maximum
```

With the emulator running you can attach a debugger.

- To attach a command line debugger, see: http://download.oracle.com/javase/6/docs/technotes/tools/windows/jdb.html

  A sample command would be:

  jdk`/bin/jdb -connect`
  `com.sun.jdi.SocketAttach:hostname=localhost,port=51307`

11.3.3 Command Line Profiling

To add profiling to an emulator session, use:

`-Xprofile:[system=<y|n>], file=`*filename*`.prof`

For example:

```
emulator.exe -Xdevice:JavaMEPhone1
-Xdescriptor:"C:\Documents and Settings\user\My Documents\Projects\Games\dist\Game
s.jad" -Xprofile:file=C:\temp\Games.prof
```

11.4 Build a Project from the Command Line

In the user interface, building a project is a single step. Behind the scenes, however, there are 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 VM. See the following topics:

- Section 11.4.1, "Check Prerequisites"
- Section 11.4.2, "Compile Class Files"

- Section 11.4.3, "Preverify Class Files"

11.4.1 Check Prerequisites

Before building and running an application from the command line, verify that the `jar` command is in your path. To find the version of the development kit, run `java -version` at the command line.

11.4.2 Compile Class Files

Use the `javac` compiler from the Java SE development kit to compile Java source files. You can use the existing Oracle Java ME SDK project directory structure. Use the `-bootclasspath` option to tell the compiler to use the MIDP APIs, and use the `-d` option to tell the compiler where to put the compiled class files.

The following example demonstrates how you might 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.

```
javac -target 1.3 -source 1.3
   -bootclasspath ..\..\lib\cldc_10.jar;..\..\lib\midp2.0.jar
   -d tmpclasses
   src\*.java
```

For more information on `javac`, consult the Java SE documentation.

11.4.3 Preverify Class Files

The next step is to preverify the class files. The `bin` directory of the Oracle Java ME SDK includes the `preverify` utility. The syntax for the preverify command is as follows:

`preverify` *`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.

```
preverify.exe
  -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.

11.5 Packaging a MIDlet Suite (JAR and JAD)

To package a MIDlet suite manually you must create a manifest file, an application JAR file, and finally, a MIDlet descriptor (also known as a Java Application Descriptor or JAD).

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. For example, a manifest might have the following contents:

```
MIDlet-1: My MIDlet, MyMIDlet.png, MyMIDlet
MIDlet-Name: MyMIDlet
MIDlet-Vendor: My Organization
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.1
MicroEdition-Profile: MIDP-2.1
```

Create a JAR file containing the manifest as well as the suite's classes 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 must 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
```

11.6 Command Line Security Features

The full spectrum of the Oracle Java ME SDK'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.

- Section 11.6.1, "Change the Default Protection Domain"

- Section 11.6.2, "Sign MIDlet Suites (jadtool)"
- Section 11.6.3, "Manage Certificates (MEKeyTool)"

11.6.1 Change the 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. Enter an appropriate security domain as described in Section 10.1, "Security Domains". For example, `-Xdomain:maximum`.

11.6.2 Sign MIDlet Suites (jadtool)

`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. For more on public key cryptography, see:

`http://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html`

`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 *installdir*`\bin`, and enter the following command:

`jadtool` *`command`*

The commands are as follows:

- `-help`

  Prints the usage instructions for `jadtool`.

- `-addcert -alias` *key_alias* `[-storepass` *password*`] [-keystore` *keystore*`] [-certnum` *number*`] [-chainnum` *number*`] [-encoding` *encoding*`] -inputjad` *filename* `-outputjad` *file*name

  Adds the certificate of the key pair from the given keystore to the JAD file.

- `-addjarsig [-jarfile` *filename*`] -keypass` *password* `-alias` *key_alias* `-storepass` *password* `[-keystore` *keystore*`] [-chainnum` *number*`] [-encoding` *encoding*`] -inputjad` *filename* `-outputjad` *filename*

  Adds a digital signature of the input JPP file to the specified output JPP file.

- `-showcert [([-certnum` *number]* `[-chainnum` *number*`]) | [-all]] [-encoding` *encoding]* `-inputjad` *filename*

  Displays information about certificates in JAD and JPP files.

The default values are as follows:

- `-encoding` - `UTF-8`
- `-jarfile` - `MIDlet-Jar-URL` property in the JAD file
- `-keystore` - `$HOME/.keystore`
- `-certnum` - `1`

- `-chainnum - 1`

11.6.3 Manage Certificates (MEKeyTool)

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 purpose of the keys is 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 (found in the \bin directory for your JDK). See:

`http://java.sun.com/javase/7/docs/technotes/tools/windows/keytool.html`

To run `MEKeyTool`, open a command prompt, change the current directory to *installdir*`\bin`, and enter the following command:

installdir`\bin\mekeytool.exe -`*command*

The command keywords follow.

The Oracle Java ME SDK contains a default ME keystore named `_main.ks,` which is located in:

installdir`\runtimes\cldc-hi\appdb`

This keystore includes all the certificates that exist in the default Java SE platform keystore that comes with the Java SE installation.

Also, each emulator instance has its own `_main.ks` file located in *userhome*`\javame-sdk\3.2\work\`*devicename*`\appdb.` If you do not specify a value for `MEkeystore`, a new key is added to the default ME key for this emulator instance.

If you do not specify a value for `-keystore`, the default keystore is used:

userhome`\keystore.ks`

- `-help`

  Prints the usage instructions for MEKeyTool.

- `-import -alias` *alias* `[-keystore` *JCEkeystore*`] [-MEkeystore` *filename*`] [-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. and the default Java Cryptography Extension keystore is *userhome*`\.keystore`.

- `-list [-MEkeystore` *filename*`]`

  Lists the keys in the ME keystore, including the owner and validity period for each.

- `-delete (-owner` *owner* `| -number` *key-number*`) [-MEkeystore` *filename*`]`

  Deletes a key from the given ME keystore with the given owner.

11.7 Generate Stubs (wscompile)

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, that 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*

Table 11–3 lists the `wscompile` options:

Table 11–3 `wscompile` ***Options***

Option	Description
`-gen`	Same as `-gen:client`
`-gen:client`	Generates client artifacts (stubs, etc.)
`-import`	Generates interfaces and value types only
`-d` *output directory*	Specifies where to place generated output files
`-f:`*features*	Enables the given features
`-g`	Generates debugging info
`-features:`*features*	Same as `-f:`*features*
`-httpproxy:`*host:port*	Specifies a HTTP proxy server (port defaults to 8080)
`-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 11–4 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 is used with a particular feature.

Table 11–4 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
`serializeinterfaces`	Turns on direct serialization of interface types	WSDL, SEI, model

Table 11–4 (Cont.) Command Supported Features (`-f`) for `wscompile`

Option	Description	Type of File
`wsi`	Enables WSI-Basic Profile features (default)	WSDL
`resolveidref`	Resolves `xsd:IDREF`	WSDL
`donotunwrap`	No unwrap.	WSDL

Examples

```
wscompile -gen -d generated config.xml
wscompile -gen -f:nounwrap -O -cldc1.1 -d generated config.xml
```

12 Logs

Oracle Java ME SDK uses the log4j logging facility to manage Device Manager and Device Instance logs.

12.1 Device Manager Logs

The device manager log is placed into:

userhome`\javame-sdk\3.2\work\`*devicename*`\device-manager.log`

Logging levels can be configured in the following XML file:

installdir`\toolkit-lib\process\device-manager\conf\log4j.xml`

A priority value for the categories `com.sun` or VM can be set to the following levels: ERROR, WARN, INFO, DEBUG, TRACE (ordered from least to most verbose).

```
<category name="com.sun">
   <priority value="DEBUG"/>
   <appender-ref ref="CONSOLE-ALL"/>
   <appender-ref ref="FILE"/>
</category>

<category name="VM">
   <priority value="INFO"/>
   <appender-ref ref="CONSOLE-ALL"/>
   <appender-ref ref="FILE"/>
</category>
```

12.2 Device Instance Logs

Each device (or emulator) instance writes its own log into its directory. See Table 6–1 to correlate the emulator number and the device name.

userhome`\javame-sdk\`*version*`\work\`*devicename*`\device.log`

`log4j.xml` controls the verbosity of the device instance logs, as described in Section 12.1, "Device Manager Logs."

13 API Support

The Oracle Java ME SDK supports many standard Application Programming Interfaces (APIs) defined through the Java Community Process (JCP) program. JCP APIs are often referred to as JSRs, named after the Java Specification Request process. JSRs that are not part of the platform are referred to as "optional packages."

The CLDC/MIDP platform is based on the *Mobile Information Device Profile and Connected Limited Device Configuration* (JSRs 118 and 139).

The IMP-NG platform is base on *Information Module Profile - Next Generation (IMP-NG)* (JSR 228).

See Table 13–1 for a full list of supported JCP APIs. The Oracle Java ME SDK provides documentation describing how certain APIs are implemented in the SDK. Many supported APIs do not require special implementation considerations, so they are not discussed in this help set. Section 13.2, "Oracle APIs" describes Oracle APIs provided to support the IMP-NG platform.

For convenience the Javadoc files that are the intellectual property of Oracle are in your installation's `\docs` directory. The remainder can be downloaded from `http://jcp.org`.

13.1 JCP APIs

Table 13–1 Supported JCP APIs

JSR, API	Name, URL
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 118, MIDP 2.0	*Mobile Information Device Profile* `http://jcp.org/en/jsr/detail?id=118`
JSR 135, MMAPI 1.1	*Mobile Media API* `http://jcp.org/en/jsr/detail?id=135`
JSR 139, CLDC 1.1	*Connected Limited Device Configuration* `http://jcp.org/en/jsr/detail?id=139`
JSR 172, Web Services	*J2ME Web Services Specification* `http://jcp.org/en/jsr/detail?id=172`

Table 13–1 (Cont.) Supported JCP APIs

JSR, API	Name, URL
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 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`
JSR 211, CHAPI	*Content Handler API* `http://jcp.org/en/jsr/detail?id=211`
JSR 226, SVG	*Scalable 2D Vector Graphics API for J2ME* `http://jcp.org/en/jsr/detail?id=226`
JSR 228, IMP-NG	*Information Module Profile - Next Generation (IMP-NG)* `http://jcp.org/en/jsr/detail?id=228`
JSR 234, AMMS	*Advanced Multimedia Supplements* `http://jcp.org/en/jsr/detail?id=234`
JSR 239	*Java Binding for OpenGL ES API* `http://jcp.org/en/jsr/detail?id=239`
JSR 256	*Mobile Sensor API* `http://jcp.org/en/jsr/detail?id=256`
JSR 257	*Contactless Communication API* `http://jcp.org/en/jsr/detail?id=257`
JSR 280, XML API	*XML API for Java ME* `http://jcp.org/en/jsr/detail?id=280`

13.2 Oracle APIs

The IMP-NG project type supports developing applications for the Oracle Java ME Embedded 3.2 runtime. The Java ME Embedded 3.2 runtime includes a number of standard JSR APIs as well as additional Oracle APIs for embedded use cases. These new APIs are:

- Device Access API

  The Device Access API provides interfaces and classes for communicating with and controlling peripheral devices.

- Logging API

  The Logging API provides a lightweight and extensible framework based on the concepts of the java.util.logging package, enabling applications to log messages in a variety of formats and using custom handlers.

- AMS API

The AMS API provides an interface to the application management capabilities of the runtime to allow authorized applications to interact with and extend the application management system.

- AccessPoint API

  The AccessPoint API is an extension to the Generic Connection Framework and enables applications to select among multiple access points if the underlying platform provides more than one data access point.

The Javadoc files for these APIs are in your installation's \docs directory.

14 JSR 75: PDA Optional Packages

The Oracle Java ME SDK 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 Oracle Java ME SDK implements the FileConnection and PIM APIs.

14.1 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 Oracle Java ME SDK emulator, the FileConnection API enables MIDlets to access files stored on your computer's hard disk.

The files that can be accessed using the FileConnection optional package are stored in the following subdirectory:

`Documents and Settings\`*user*`\javame-sdk\3.2\work\`*devicename*`\appdb\filesystem`

For example, the `JavaMEPhone1` emulator instance comes with a root directory installed named `root1`. Each subdirectory of `filesystem` is called a *root*. The Oracle Java ME SDK provides a mechanism for managing roots.

While the emulator is running, choose Device > File Connection. The External Events Generator opens with the File Connection tab selected.

In the File Connection panel you can mount, unmount, or delete filesystem roots. Mounted roots are displayed in the top list, and unmounted roots are moved to the bottom list. Mounted root directories and their subdirectories are available to applications using the FileConnection API. Unmounted roots can be remounted in the future.

- To add a new empty filesystem root directory, click Mount Empty and fill in a name for the directory.
- To mount a copy of an existing directory, click Mount Copy, and browse to choose a directory you want to copy. When the File System Root Entry dialog opens, enter the name for this root. A deep copy of the selected directory is placed into the emulator's filesystem with the specified root name.

- To make a directory inaccessible to the FileConnection API, select it in the list and click Unmount. The selected root is unmounted and moved to the Unmounted roots list.
- To completely remove a mounted directory, select it and click Unmount & Delete.
- To remount an unmounted directory, select it and click Remount. The root is moved to the mounted roots list.
- To delete an unmounted directory, select it and click Delete. The selected root is removed from the list.

14.2 PIM API

The Oracle Java ME SDK emulator stores contact, calendar, and to-do information in standard files on your desktop computer's hard disk. All information is stored in:

`Documents and Settings\`*user*`\javame-sdk\3.2\work\`*devicename*`\appdb\PIM`

Each device instance has its own data. Lists are stored in subdirectories of the `contacts`, `events`, and `todo` directories. For example, a contact list called Contacts is contained in *installdir*`\appdb\PIM\contacts\Contacts`.

Inside the list directory, items are stored in vCard (`.vcf`) or vCalendar (`.vcs`) 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.

14.3 Running PDAPDemo

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

14.3.1 Browsing Files

The default emulators have one directory, `root1`. This directory is located at:

userhome`\javame-sdk\3.2\work\`*devicename*`\appdb\filesystem\root1`

For test purposes, copy files or even directories into root1. You can also add other directories at the same level as root1.

Now open and run the `PDAPDemo` project.

- Launch the `FileBrowser` MIDlet. You see a directory listing, and you can browse through the directories and files you have placed there.
- Select a directory and press the View soft button to enter it.
- Using the Menu commands you can view a file or see its properties. Try selecting the file and choosing Properties or View from the menu.

  You can view the content of text files in the browser.
- Try using the External Events Generator to unmount and mount directories. Unmounted directories are not visible in the application running on the emulator.

14.3.2 The PIM API

The JSR75 PIM APIs example demonstrates how to access personal information, such as contact lists, calendars, and to-do lists.

- After you launch the example, choose a type of list from the main menu.

In this example 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.

- After you have 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 have run the example, the list might be empty.
- To add an item, choose New from the menu. The application prompts you for a Formatted Name for the item.

  To add more data fields to this item choose the menu item Add Field. You see a list of field names. Pick as many as you like. You can fill in the fields at any time.
- To save the new item, choose Commit from the menu.

  To return to the list, choose 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 this directory:

  `userhome\javame-sdk\3.2\work\device-name\appdb\PIM`

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.

15 JSR 82: Bluetooth and OBEX Support

This chapter describes how the Oracle Java ME SDK implements the Bluetooth and OBEX APIs.

The Oracle Java ME SDK 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 Oracle Java ME SDK emulator enables you to develop and test applications that use Bluetooth without having actual Bluetooth hardware. The SDK 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 Section 15.2, "Running the Bluetooth Demo."

- The OBEX API allows applications to use the Object Exchange (OBEX) protocol over Bluetooth or other communication channels.

  The Oracle Java ME SDK implements OBEX transfer over simulated Bluetooth and TCP connections.

  For an example, see Section 15.3, "Running the OBEX Demo."

15.1 Setting OBEX and Bluetooth Properties

The Oracle Java ME SDK enables you to configure the Bluetooth and OBEX simulation environment. Because the simulation requires a sender and receiver, Bluetooth settings are configured separately for each device. Follow these steps to set device properties.

1. In the device selector double-check on a CLDC device.

   The device properties are displayed in the Properties pane.

2. Scroll down to see the Bluetooth and OBEX properties. When you click a property a description is shown in the description area.

The System Properties 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.

- **Bluetooth Enabled**

  Enable or disable Bluetooth functionality

- **Bluetooth Address**

  The Bluetooth address of this device.

- **Friendly Name**

  Set the friendly name of the device
- **bluetooth.sd.trans.max**

  The maximum number of concurrent service discovery transactions. The default is 8.
- **bluetooth.sd.attr.retrievable.max**

  The maximum number of service attributes to be retrieved per service record.The default is 16.
- **bluetooth.master.switch**

  Enable/disable a master/slave switch. Enabled by default.
- **bluetooth.l2cap.receiveMTU.max**

  The maximum ReceiveMTU size in bytes supported in L2CAP. This is the maximum payload size this connection can accept.

  The default value is 672.
- **OBEX Maximum Packet Length**

  The default is 4096 bytes.

  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.
- Device is discoverable

  Enabled by default.
- Authentication is enabled

  Enabled by default.
- Encryption is enabled

  Enabled by default.
- Authorization is enabled

  Enabled by default.

15.2 Running the Bluetooth Demo

This application contains MIDlets that demonstrate the use of JSR 82's Bluetooth API. It shows how images can be transferred between devices using Bluetooth.

You must run two emulator instances to see this process, and each device must have a different phone number.

1. Use JavaMEPhone1 to launch Bluetooth Demo, then launch Bluetooth Demo on JavaMEPhone2.
2. The demo gives you a choice of Server or Client.
3. On the first emulator, highlight Server and use the right soft button to choose OK.

   The server starts and displays a list of images. At the beginning, none of the images are available on the Bluetooth network.

   Select the image you want to make available.

Press Publish image (the right soft button). The icon color changes from purple to green, indicating it is published.

4. On the second emulator running the Bluetooth Demo, highlight Client and choose OK. The MIDlet displays "Ready for images search". Click the Find soft button. The MIDlet finds the other emulator and gets a list of published images. Select one from the list and choose Load.

   - 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.

15.3 Running the OBEX Demo

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.

1. Launch two instances of the emulator. One listens for incoming connections, while the other attempts to send an image.

   For example, right-click ObexDemo, select Run With and choose the device JavaMEPhone1. Repeat and choose JavaMEPhone2.

2. In the first emulator, choose Receive Image. (Depending on your security level, the application warns that an OBEX connection allows other devices to talk to yours and asks, "Is it OK to make the connection?" Choose Yes.) Choose Start to run the application. The listener emulator displays a screen reading "Waiting for connection".

3. In the second emulator (the sender), choose Send Image and press the Start soft key. Select an image from the list and choose Send. (Depending on your security level, the application warns that the demo wants to make an outgoing client connection, and asks if it is OK. Choose Yes.) The Send Image utility uploads the image.

4. In the listening emulator, the utility displays information about the incoming image and asks "Would you like to receive it?" Choose yes.

   The image you selected is transferred over the simulated infrared link and displayed on the first emulator.

16 JSR 135: Mobile Media API Support

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 given the media formats, protocols, and features supported by various devices. See the following topics:

For information on programming with MMAPI, see the following articles:

Mobile Media API Overview:
http://www.oracle.com/technetwork/systems/mmapi-overview-156523.html

The J2ME Mobile Media API: http://www.jcp.org/en/jsr/detail?id=135

16.1 Media Types

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

MIME Type	Description
audio/amr*	Adaptive Multi-Rate Narrow Band
audio/midi	MIDI files
audio/mpeg*	MP3 files
audio/mp4*	MP4 Audio 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/3gpp*	Third generation mobile broadband with video

MIME Type	Description
video/mpeg*	MPEG video
video/mp4*	MP4 video files
video/avi*	Video capture emulation and Audio-Video Interleaved files

In the previous table, an asterisk (*) indicates a media type that requires corresponding DirectShow filters to be installed on your system. For example, MP3 support might require an MP3 Splitter and an MP3 Decoder (these might be two separate DirectShow filters, or they might be combined in one filter). You can use any appropriate filter, but Java ME SDK 3.2 has only been tested with filters from the K-Lite Mega Codec Pack 4.8.0. If no appropriate DirectShow filters are found on your system, JSR 135 Player creation for the media type might fail.

16.1.1 Media Capture

The Oracle Java ME SDK emulator supports audio and video capture. Audio capture is supported 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.

16.2 MMAPI MIDlet Behavior

MIDlets have a lifecycle 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 Oracle Java ME SDK 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.

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

16.3 Ring Tones

MMAPI plays ring tones, as demonstrated in Section 16.5.1, "Simple Tones" and Section 16.5.2, "Simple Player." The ring tone formats mentioned here are in common use. You can download ring tones or create your own.

16.3.1 Download Ring Tones

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

- `http://www.convertyourtone.com/`
- `http://www.phonezoo.com`

16.3.2 Ring Tone Formats

This section provides samples of several formats

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

  `http://en.wikipedia.org/wiki/Ring_Tone_Transfer_Language`

- 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 Minuet 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

  Beethoven's 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

  Beethoven's 9th symphony:

  ```
  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

Beethoven's 9th symphony:

```
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#****
```

16.4 Running AudioDemo

Demonstrates audio capabilities, including mixing and playing audio with an animation. Select a MIDlet from the list, and from the menu, select 1, Launch.

- Audio Player.

  Select a sound clip and press the Play soft button. Click Back to return to the list of clips.

- Bouncing Ball. Select No Background and press the Play soft button. Two balls randomly bounce in the screen, emitting a tone whenever they contact a wall.

  Wave background, tone seq background, and MIDI background play the same two-ball audio visual sequence with the additional audio background.

- Mix Demo shows that different audio formats can play simultaneously. Select a MIDlet and press the Play soft button.

  Tone+Wav - The audio clip starts playing and the Tone soft button is displayed. Press the Tone button to hear a tone playing over the original audio clip.

  Tone+ToneSeq - The audio clip starts playing and the Tone soft button is displayed. Press the Tone button to hear a tone playing over the original audio clip.

  ToneSeq+Wav - The tone sequence and the wav sequence play simultaneously. Press the Pause soft button to interrupt, and press Play to resume.

16.5 Running MMAPIDemos

The MMAPIDemos application contains four MIDlets that showcase the SDK's multimedia capabilities:

16.5.1 Simple Tones

Simple Tones demonstrates how to use interactive synthetic tones. Select a sample, 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.

16.5.2 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 00:01.04. 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 ring tone 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.
- AMR Narrow band. Plays an Adaptive Multi-rate narrow band file. This sample requires an AMR codec. This sample was tested with the K-Lite Mega Codec Pack 4.8.0. This codec is freely downloadable.
- 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. For example, on a device equipped with a camera.
- `[enter URL]` Plays back media files from arbitrary HTTP servers. Type a valid URL 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 menu lists commands that control media playback.

The first menu item, Quick Help, displays a list of commands and actions mapped to keypad buttons. Some actions are not applicable for every media type.

The remaining menu items vary depending on the media type. Some actions, such as Rate, open a control with which you can arbitrarily change the playback. Click Back to return to the player screen and see or hear your changes.

16.5.3 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 shows 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 in a Form. A simple indicator shows the progress through the animation.
- **Animated GIF - Canvas [jar]** shows an animated GIF in a Canvas. A simple indicator shows the progress through 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 is placed beneath the video capture for comparison.
- **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.

16.5.4 Pausing Audio Test

This test MIDlet demonstrates the proper use of `pauseApp()`and the alternative.

WARNING: Do not run the memory monitor while using this demo.

In the Well-Behaved case suspending uses `pauseApp()` to close the player and remembers the length of time the audio file played. When the player resumes, it starts playing the audio file at the point that it was suspended.

In the Not Well-Behaved case the player is stopped instead of suspended. When the player is restarted the audio file plays from the beginning.

Test the two cases as follows:

- Run MMAPIDemos, and launch Pausing Audio Test.

  The music starts playing. The initial value of Current State is Well-Behaved.
- Select Application > Suspend (or F5), to pause the music.
- Select Application > Resume (or F6) to resume playing the audio clip from the stopping point.
- Click the Misbehave soft key.
- Select Application > Suspend (or F5), to stop the music.

- Select Application > Resume (or F6). The player restarts but the clip plays from the beginning.

17 JSR 177: Smart Card Security (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 Oracle Java ME SDK emulator fully supports SATSA. This topic describes how you can use the Oracle Java ME SDK 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://download.oracle.com/javame/config/cldc/opt-pkgs/api/security/satsa-dg`.

If you must develop your own Java Card applications, download the Java Card Development Kit, available at `http://www.oracle.com/technetwork/java/javacard/overview/index.html`. This kit is for Windows.

17.1 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 must specify a slot and a card application.

The Oracle Java ME SDK 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 socket protocol. 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 Oracle Java ME SDK 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.

The default card emulator host name is localhost, and the default ports are 9025 for slot 0 and 9026 for slot 1. These port defaults are a property of the device. To change the defaults in the user interface, right click on the device in the Device Selector, and select Properties. By default the Properties window is docked on the upper right of the Java ME SDK interface.

You can also change the port values in the device's property file found at:

userhome`\javame-sdk\3.2\work\`devicename

Edit the `device.properties` file and modify this line:

```
runtime.internal.com.sun.io.j2me.apdu.hostsandports =
localhost:9025,localhost:9026
```

17.2 Java Card Platform Simulator (cref)

The Oracle Java ME SDK includes the Java Card Platform Simulator, which you can use to simulate smart cards in the Oracle Java ME SDK emulator's slots. The Java Card Platform Simulator is found in the following location:

installdir`\bin\cref.exe`

Going forward, this document refers to the executable as cref. The basic procedure for testing SATSA applications with the Oracle Java ME SDK is as follows:

1. Start `cref` with a Java Card platform application.
2. Start the emulator.

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

   It is important to start `cref` with the same port number as one of the slot port numbers you specified in the Oracle Java ME SDK preferences.

For example, to run `cref` on port 9025 with a prebuilt memory image, use a command line similar to this:

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

The Oracle Java ME SDK includes a demonstration application, `Mohair`, which illustrates how to use SATSA. For detailed instructions on running `Mohair`, see Section 17.4.4, "MohairMIDlet."

17.3 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*\devicename\appdb directory. For example, an access control file for slot 0 might be:

`Documents and Settings\`*user*`\javame-sdk\3.2\work\`*devicename*`\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.

17.3.1 Specifying PIN Properties

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

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

17.3.2 Specifying Application Permissions

Application permissions are defined in access control file (`acf`) records. The record format is as follows:

```
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 signature
                   ...
                 }
         }
         ...
         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.

17.3.3 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
    }
  }
}
```

17.4 Running 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 SDK, as discussed in Section 17.2, "Java Card Platform Simulator (cref)."

Note: For the demo to work this project must reside in the Java ME SDK installation's \apps subdirectory. You must create the apps directory yourself.

1. Create an `\apps` directory in your Java ME SDK platform installation. For example, `C:\Java_ME_platform_SDK_3.2\apps`.
2. In Eclipse, select File > Switch Workspace > Other.

   Browse to the `\apps` directory, and click OK.
3. Select File > New >Project... and in the New Project window, select Examples > Java ME SDK 3.2 > Java ME Sample Applications, and click Next.

   Choose SATSADemos.
4. Click Finish.
5. Run SATSADemos.

You are ready to run the MIDlets in the SATSADemos MIDlet suite.

17.4.1 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. See Section 17.2, "Java Card Platform Simulator (cref)."

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.

1. Start up two instances of `cref`, one for each simulated card slot (assuming the current directory is the SDK installation directory):

   ```
   start installdir\bin\cref -p 9025 -i installdir\apps\SATSADemos\demo2.eeprom

   start installdir\bin\cref -p 9026 -i installdir\apps\SATSADemos\demo2.eeprom
   ```
2. When you have the two smart card simulators running, run SATSADemos. Select APDUMIDlet, select the Menu soft key and select Launch (1). Press Go when prompted.

   The emulator screen displays the process of exchanging APDUs between eeproms.

17.4.2 SATMIDlet

`SATMIDlet` demonstrates smart card communication with a slight variation on APDU communication.

1. To set up the simulated smart card, use `cref`, very much like you did for `APDUMIDlet`. This time you do not have to specify a port number, and the memory image is different:

   ```
   start installdir\bin\cref -i installdir\apps\SATSADemos\sat.eeprom
   ```
2. When you have the smart card simulator running, run SATSADemos. Select SATMIDlet, select the Menu soft key and select Launch (1). Press Go when prompted.

   The emulator screen displays the process of sending envelopes over a SAT connection.

17.4.3 CryptoMIDlet

`CryptoMIDlet` demonstrates the general cryptographic features of SATSA. It does not interact with a smart card in any way. Choose the MIDLet and launch it to see the cryptography results. Use the up and down navigation keys to scroll the display.

17.4.4 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 must start `cref` with the right memory image to prepare for the connection from `MohairMIDlet`.

1. Start `cref` from the SDK installation directory:

   start *installdir*`\bin\cref -p 9025 -i` *installdir*`\apps\SATSADemos\pki.eeprom`

2. In the emulator, select Find Slots. After you see the slots found, select the Back soft key.
3. Select SATSA-PKI Sign test. The following confirmation message appears:

   ```
   This certificate will be used: Certificate two
   ```

   Select the OK soft key.

4. For PIN 1, type: `1234`

   Select the OK from the menu. The following confirmation message appears:

   ```
   This string will be signed: JSR 177 Approved
   ```

5. Select the OK soft key. The following confirmation message appears:

   ```
   This certificate will be used: Certificate one
   ```

   Select the OK soft key.

6. For non repudiation key 1 PIN, type: `2345`

   Select the soft menu and choose OK (option 2). The sign test is complete.

17.5 Running 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.

Like SATSADemos, this application must also reside in the Java ME SDK installation's `\apps` subdirectory, and `\apps` must be current workspace. See the introduction to Section 17.4, "Running SATSADemos." Also, as with some of the MIDlets in `SATSADemos`, you must start `cref` with an appropriate memory image.

1. Right-click on the project, select Properties and choose the Java ME category. View the configuration to determine the target device.
2. In the Device Selector, click the target device to see its properties. Expand the General category and locate the Security Domain property. If the value is not set to Maximum, choose Maximum from the drop-down list.
3. Start `cref` from the SDK installation directory as follows:

start *installdir*`\bin\cref -p 9025 -i` *installdir*`\apps\SATSADemos\demo2.eeprom`

4. Now run `JCRMIMIDlet` to see how your application can communicate with a distributed object on the card.

   The emulator screen displays the process of exchanging APDUs between eeproms.

18 JSR 179: Location API Support

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 Oracle Java ME SDK includes a simulated location provider. You can use the emulator's External Events Generator 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.

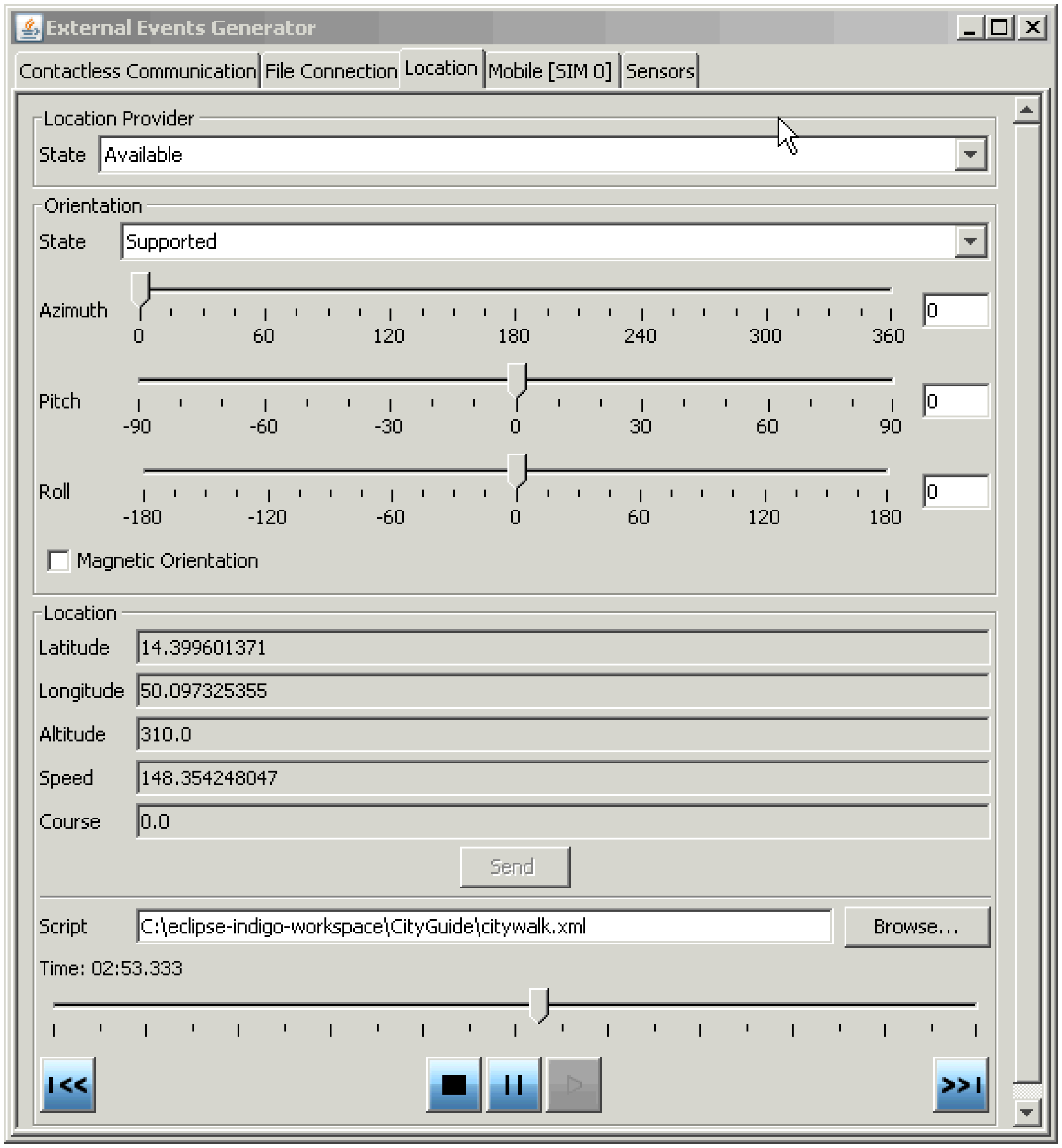
18.1 Setting the Emulator's Location at Runtime

You can specify the simulated location of the emulator while it is running. In the emulator choose Device > Location. This raises the external events generator with the Location tab selected.

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 Oracle Java ME SDK 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>
```



Location Provider State: Available

Orientation State: Supported

Azimuth: 0

Pitch: 0

Roll: 0

Magnetic Orientation: ☐

Location

Latitude: 14.399601371

Longitude: 50.097325355

Altitude: 310.0

Speed: 148.354248047

Course: 0.0

Script: C:\eclipse-indigo-workspace\CityGuide\citywalk.xml

Time: 02:53.333

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 check box 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.

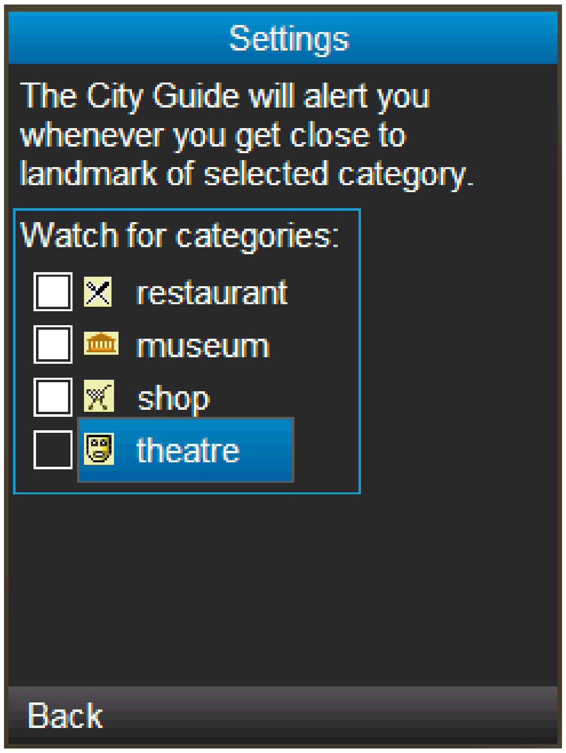
18.2 Running the CityGuide Sample Project

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. This demo gets the walker's location from an XML script named `citywalk.xml` (the event file) that submits the device location information.

Because location prompts occur frequently, it is best to run this demonstration in manufacturer (trusted) mode, as explained in "Section 10.1, "Security Domains."" In the user interface, right-click on your project and select the Running category. Select Specify the Security Domain, and select manufacturer or maximum.

1. Open and run the CityGuide project. In the emulator, launch the CityGuide MIDlet. The map page opens.
2. By default the display shows icons for four types of landmarks: restaurants, museums, shops, and theaters.

   To adjust the landmark display (this is optional), open the soft menu and choose the Settings command. Use the navigation keys to highlight a category, then use Select to check or uncheck an item. In the default skin the item is selected when the square is filled with white.


Settings

The City Guide will alert you whenever you get close to landmark of selected category.

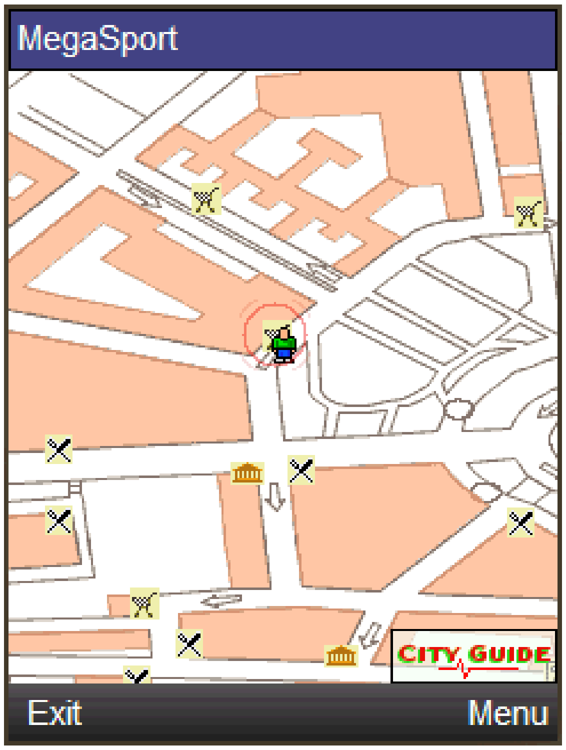
Watch for categories:

- ☐ restaurant
- ☐ museum
- ☐ shop
- ☐ theatre

Back

3. In the emulator, choose Device > Location. On the Location tab, click the Browse button. Select the event file from the directory containing the Citywalk application.

   The player buttons at the bottom of the window are now active. Press the green play button (right-pointing triangle) to run the script.
4. When you are near a landmark its name appears at the top of the map. Open the soft menu and choose the Detail command to see more information.


MegaSport
CITY GUIDE
Exit
Menu

19 JSR 205: Wireless Messaging

The Oracle Java ME SDK 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 all this and support for Multimedia Message Service (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 tool for testing WMA applications.

Many of the tasks in this topic can also be accomplished from the command line. See Section 19.3, "Running WMA Tool."

19.1 Using the WMA Console to Send and Receive Messages

The WMA console is a tool that enables you to send messages to and receive messages from applications that use JSR 205. You can, for example, use the WMA console to send SMS messages to a MIDlet running on the emulator.

19.1.1 Launching the WMA Console

Be sure to select the Java ME perspective. Select Window > Open Perspective > Other > Java ME.

To launch the WMA console, select Run > WMA Console. Messages can be sent from the WMA Console to an emulator instance.

The console opens as a tab in the Eclipse documents area. The console phone number is displayed as part of the WMA Console tab label (for example, 987654321).

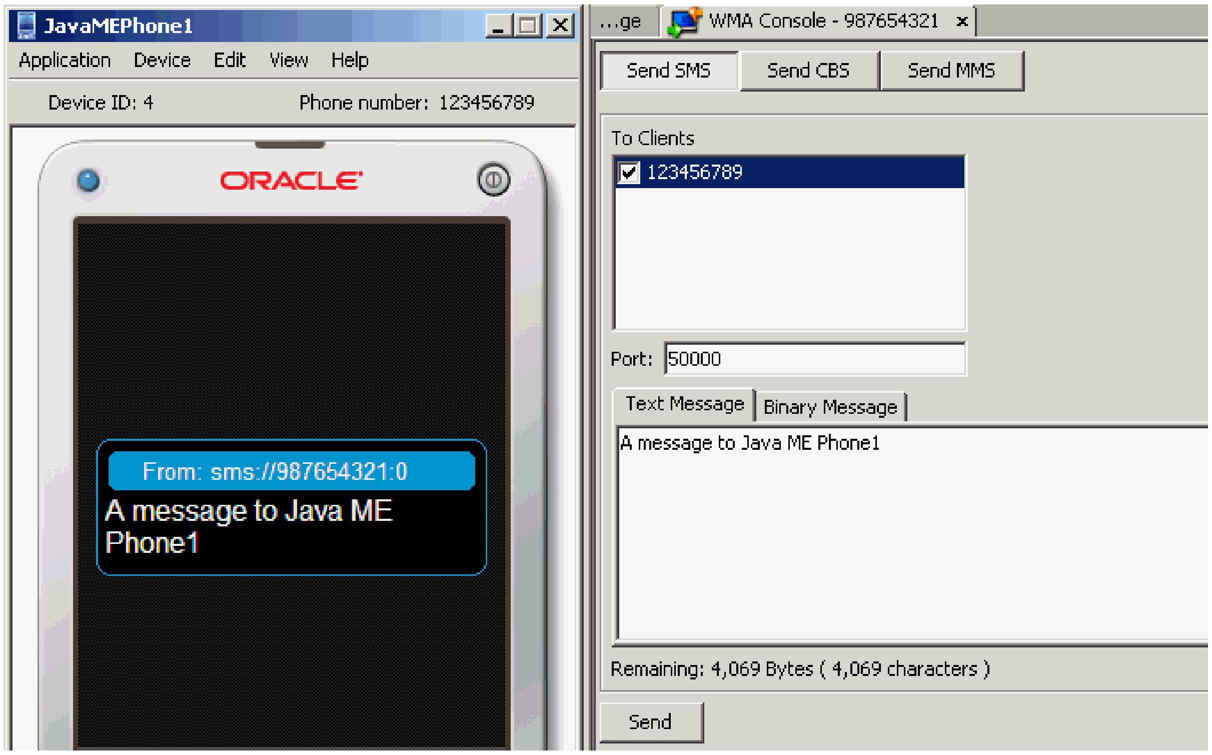
The WMA console phone number is an editable CLDC property. To edit a property, follow these steps:

1. In the Device Selector, click the Oracle Java ME Platform SDK 3.2 node.
2. Type a new value in the WMA Console Phone Number 1 field.

   If the number is available it is assigned to the console immediately. If the number is in use it is assigned to the console the next time you restart the IDE.

19.1.2 WMA Console Interface

To open the WMA Output window, select Run > WMA Console Output. This window displays messages received from an emulator.



19.1.3 Emulator Phone Numbers

Each instance of the emulator has a simulated phone number that is shown in the emulator window. The phone numbers are important because they are used as addresses for WMA messages. The phone number is a device property and it can be changed. In the device selector, right-click a device and view its properties.

19.1.4 Sending a Text or Binary SMS Message

To send a text SMS message, click Send SMS.

- The To Clients window automatically lists the phone numbers of all running emulator instances. Select one or more destinations and enter a port number (the default is 50000, as described in Section 19.2.1, "WMADemo Push Registry Values."
- To send a text message, select the Text Message tab, type your message and click Send.
- To send the contents of a file as a binary message, click the Binary Message tab. Type in the path of a file directly, or click Browse to open a file chooser.

Note: The maximum message length for text and binary messages is 4096 bytes.

To try this yourself see Section 19.2.2, "Sending SMS Messages From WMA Console to an Emulator and Back."

19.1.5 Sending Text or Binary CBS Messages

Sending CBS messages is similar to "Sending a Text or Binary SMS Message" except that recipients are unnecessary because it is a broadcast.

To send a text or binary CBS message, click Send CBS in the WMA console. Specify a message identifier (see Section 19.2.1, "WMADemo Push Registry Values") and enter the text or binary content of your message. The maximum message length for text and binary messages is 4096 bytes.

Note: The emulator displays only the first 160 symbols of a received CBS message.

19.1.6 Sending MMS Messages

MMS messages consist of one or more files, usually images or sounds. An 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 Header and Parts tabs.

- The header tab addresses the message.

  The To area automatically lists one of the phone numbers from the running emulator instances, and you can click the Add button to select other available phone numbers from the drop-down list.

  To remove a recipient, first select its line, then click Remove.

  When a recipient is removed it must be added back manually. Click the Add button and a new line is added to the recipient table.

- To add optional media files (Parts) to the message, click the Parts tab and click Add. The maximum message length for text and binary messages is 4096 bytes.

  Most media files have information to fill the Content Location, Content ID, Mime-Type (text/plain for simple MMS), and Encoding fields, but you can edit these fields as well. The default ID for the demo is `example.mms.MMSDemo` (see Section 19.2.1, "WMADemo Push Registry Values").

  To remove a part, select it and press Remove.

To try this yourself, see Section 19.2.3, "Sending MMS Messages from WMA Console to an Emulator."

19.1.7 Receiving Messages in the WMA Console

To start the WMA console, select Run > WMA Console. The WMA console window has its own phone number displayed on the WMA Console tab. You can send messages from your applications running on the emulator to the WMA console.

Received messages are displayed in the WMA output window.

19.2 Running WMADemo

The WMADemo sample project shows how to send and receive SMS, CBS, and MMS messages. Messages can be exchanged between emulator instances and can be generated or received using the WMA console utility.

19.2.1 WMADemo Push Registry Values

The push registry determines how the demo establishes certain types of connections. This information is set in the Application Descriptor. To view it, right-click on the

WMA Demo project and select Properties. In the Properties window, go to Application Description>Optional>Push Registry.

- For SMS messages the port number is 50000.
- For CBS Messages, the Message Identifier is 50001.
- For MMS messages, the application ID is `example.mms.MMSDemo`.

19.2.2 Sending SMS Messages From WMA Console to an Emulator and Back

In this demo you send messages between the WMA Console and the client demo application running on the emulator. Using the WMA console to send messages to the emulator exercises the push registry.

1. To launch the WMA console, select Run > WMA Console. To open the WMA Output window, select Run > WMA Console Output. The WMADemo should be running in the emulator.
2. Click on the Send SMS button in the WMA console window.

   Choose the number that corresponds to the emulator. Typically you check the box in front of 123456789. If you are not sure what number the emulator is using, look for a number above the emulator screen.

   Fill in a port number of 50000. This is required because the demo waits for the SMS on that port.

   Type your text message in the Message field and click Send.
3. The emulator asks if it is OK if the `WMADemo` interrupts and if it can be started. You might receive several permission requests based on your firewall settings.

   Choose Yes. The `SMSReceive` MIDlet is launched and immediately displays the incoming SMS message.
4. To type a return message, press the Reply soft button. Type a message and select Send from the menu. You might be asked to give permission because there is a cost to your phone number. In the IDE, look in the WMA Output Window to confirm that your reply has been received. (The output window is typically displayed below the WMA Console. Be sure to click the WMA Output Window tab.)

19.2.3 Sending MMS Messages from WMA Console to an Emulator

To send an MMS message from the WMA console to the emulator, follow these steps:

1. From the WMADemo home screen, choose MMS Receive. The emulator displays: "MMS Receive" and the message "Waiting for MMS on applicationID example.mms.MMSDemo..."
2. In the WMA console, click Send MMS to open the MMS composition window. The Header tab is open by default. Supply any message subject, the application ID `example.mms.MMSDemo`, and the telephone number of the running emulator. That number is displayed to the right of the To field by default.

   The Address Type field on the left is a drop-down list from which you can choose To, Cc or Bcc.

   If you do not see the number of a running emulator, click the Add button to add it to the Address field. When you have listed multiple numbers, the Address field becomes a drop-down list.

3. Click the Parts tab. The WMA console enables you to select files to send as parts of the MMS message. Click Add and use the file browser to find the file you want to send. Click OK.
4. Click Send to send the message.

   The image and its information are displayed in the emulator.

19.3 Running WMA Tool

WMA Tool is the command line version of the WMA Console. To send and receive SMS, CBS, and MMS messages from the command line, run:

```
installdir\bin\wma-tool <command> [options]
```

The device manager must be running before you launch `wma-tool`.

When the tool is started, it outputs the phone number it is using.

Command

Each protocol has send and receive commands. The requested command is passed to the tool as a first argument. Possibilities are:

- `receive`
- `smsreceive` - receives SMS messages
- `cbsreceive` - receives CBS messages
- `mmsreceive` - receives MMS messages
- `smssend` - sends SMS message
- `cbssend` - sends CBS message
- `mmssend` - sends MMS message

The *send commands send the specified message and exit. The *receive commands print incoming messages until they are explicitly stopped.

Options

`-o` *outputDir*. Store binary contents to *outputDir*.

`-t` *timeout*. Non-interactive mode, waits the number of *timeout* seconds for messages.

`-f` Store text contents as files instead of printing them.

`-q` Quiet mode.

19.3.1 smsreceive, cbsreceive, and mmsreceive

The syntax for receiving a message is basically the same for all three protocols.

```
smsreceive [-o outputDir] [-t timeout] [-q]
```

```
cbsreceive [-o outputDir] [-t timeout] [-q]
```

```
mmsreceive [-o outputDir] [-t timeout] [-q]
```

19.3.2 smssend

```
wma-tool smssend target_phone target_port message_content
```

- *target_phone*

Phone number of the target phone. Mandatory first argument.

- *target_port*

  Port of the target phone. Mandatory second argument.

- *message_content*

  Mandatory third argument. Can have one of these two forms:

  - `text`: text of the text message
  - `-f` *file*: sends content of the specified file as a binary message.

  Example:

  ```
  wma-tool smssend 123456789 50000 "smssend message from wma-tool"
  ```

19.3.3 cbssend

wma-tool cbssend *message_id* *message_content*

- *message_id*

  ID of the message. Mandatory first argument.

- *message_content*

  Mandatory second argument. Can have one of these two forms:

  - `text`: text of the text message
  - `-f` *file*: sends content of the specified file as a binary message.

  Example:

  ```
  wma-tool cbssend 50001 "cbssend message from wma-tool"
  ```

19.3.4 mmssend

```
wma-tool mmssend applicationId subject
     [-to <targetphone>]* [-cc <target phone>]* [-bcc <target phone>]*
     [-part { <part_from_file> | <part_from_text> } ]*
```

Each part is defined by `name=`*value* pairs delimited by a semicolon ";" separator.

Part Variables

To create *part_from_file*, define the following variables.

Note: The file and the mimeType must be separated by a semicolon.

- `file`

  File to send as a message part.

- `mimeType`

  Mime type of the file.

To create *part_from_text*, define the following variables:

- `text`

  Text to send as a message part. `mimeType` is set to `text/plain`.

- `-to` *target_phone*

"to" target phone number. You can use any number of these options.

- `-cc` *target_phone*

  "cc" target phone number. You can use any number of these options.

- `-bcc` *target_phone*

  "bcc" target phone number. You can use any number of these options.

Part from Text Options

Separate options with semicolons. For example:

- `-part contentId=`*content ID*; `encoding=`*encoding*; `text=`*text*

  Appends text part to the message. You can use any number of these options. Contains the following options:

  - *content ID*: content ID of this message part
  - *encoding*: Sent text encoding. Only relevant for "text/plain". Mime type defaults to UTF8.

Part from File Options

`-part mimeType=`*mime type*; `contentId=`*content ID*`;  file=`*file name*

- Appends binary part to the message with content loaded from the given file. You can use any number of these options.

  Separate the options with a semicolon.

  - *content id*: content ID of this message part
  - *mime type*: mime type of this message part
  - *file name*: file with content of this message part
  - *fileEncoding*: Encoding of text in the file, only relevant for "text/plain", only applies if the file argument is present. Defaults to the value of the `encoding` variable.

Example:

```
wma-tool mmssend example.mms.MMSDemo MySubject -to 123456789 -part
file=Duke.png;mimeType=image/png
```

20 JSR 184: Mobile 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.

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.

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

20.1 Choosing a Graphics Mode

Applications are free to use whichever approach is most appropriate or to use a combination of the retained mode and immediate mode APIs.

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.

20.1.1 Immediate Mode

Immediate mode is appropriate for applications that generate 3D graphics content algorithmically, such as 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.

20.1.2 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.

20.2 Quality Versus 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 simplifies the implementation's task because it reduces the number of calculations required to show a scene.

In general, 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.

20.3 Content for Mobile 3D Graphics

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.

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.

20.4 Running Demo3D Samples

Demo3D contains MIDlets that demonstrate JSR 184 features.

Go to File > New > Project and in the Categories window select Examples > Java ME SDK 3.2 and click on Java ME Sample Applications. Click Next. In the next screen choose Demo3D and click Finish.

20.4.1 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.

The keypad buttons in Table 20–1 provide control over the game.

Table 20–1 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 can be found at:

```
projects\Demo3D\src\com\superscape\m3g\wtksamples\life3d\Life3D.java
```

Where projects is the directory you are using to store your Eclipse projects.

20.4.2 RetainedMode

The `RetainedMode` MIDlet plays a scene file that shows a skateboarder in an endless loop. The source code is found at:
projects\Demo3D\src\com\superscape\m3g\wtksamples\retainedmode

20.4.3 PogoRoo

`PogoRoo` displays a kangaroo bouncing up and down on a pogo stick. To steer the kangaroo, use the arrow keys. Press up to go forward, down to go backward, and left and right to change direction. Try holding down the key to see an effect. The source code is found at:

projects\Demo3D\src\com\superscape\m3g\wtksamples\pogoroo

21 JSR 211: Content Handler API (CHAPI)

JSR 211 defines the Content Handler API (CHAPI). The basic concept 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.

21.1 Using Content Handlers

In the Oracle Java ME SDK Content Handlers are integrated in an Eclipse project as application descriptors. Content Handlers you define are packaged with the application.

21.2 Running the CHAPIDemo Content Browser

This demo is a content browser that takes advantage of the content handler registry. It enables you to view different types of content from different sources.

Note: For the demo to work this project must reside in the Java ME SDK installation's `\apps` subdirectory. You must create this directory yourself.

1. If your workspace is the Java ME SDK installation's `\apps` subdirectory, continue to step 2. If not, follow these steps to create the workspace:
   - Create an `\apps` directory in your Java ME SDK platform installation. For example, `C:\Java_ME_platform_SDK_3.2\apps`.
   - In Eclipse, select File > Switch Workspace > Other.
   - Browse to the `\apps` directory, and click OK.
2. Select File > New >Project... and in the New Project window, select Examples > Java ME SDK 3.2 > Java ME Sample Applications, and click Next.

   Choose CHAPIDemo.
3. Run the project.
4. On the Favorite Links page, choose CHAPI Demo.
5. Press Select or choose the menu soft button and choose Go.

   The Text Viewer displays a Media Player URL and links to various media files.
6. Install the Media Player to view media.

- Click the URL `http:handlers/MediaHandler.jad`, or, use arrow keys to highlight the URL and from Menu, select Go.
- The application asks, "Are you sure you want to install Media Handler?" Click the Install soft key.

  An authorization Information screen is displayed.
- Press the Install soft key.

  The installation is confirmed. When the installation completes, you are returned to the Text Viewer. The Media Handler is shown as a separate application in the AMS.

7. Select and view the different image, video, audio and text URLs.

Click on a link to open that media in the viewer, or, use arrows to highlight the link, then select Go from the soft menu.

Select the Back soft key to return to the Text Viewer.

22 JSR 226: Scalable 2D Vector Graphics

JSR 226, Scalable 2D Vector Graphics for J2ME, supports rendering sophisticated and interactive 2D 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 Scalable Vector Graphics (SVG) 1.1 specification (available at `http://www.w3.org/TR/SVG11/`) defines a language for describing two-dimensional graphics in XML.

SVG Tiny (SVGT) is a subset of SVG that is appropriate for small devices such as mobile phones. See `http://www.w3.org/TR/SVGMobile/`. SVGT 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.

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 animation, as illustrated in Section 22.1.3, "Play SVG Animation." The other is to repeatedly modify the SVG image parameters (such as color or position), through API calls.

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

- **Inkscape:** `http://inkscape.org`
- **Adobe Illustrator:** `http://www.adobe.com/products/illustrator/main.html`

22.1 Running SVGDemo

This project contains MIDlets that demonstrate different ways to load manipulate, render, and play SVG content.

22.1.1 SVG Browser

The SVGBrowser MIDlet displays SVG files residing in the phone file system. Before running this demo, place an SVG file in your device skin's file structure. The default location is:

userhome\javame-sdk\3.2\work\device\appdb\filesystem\root1

For your device location, see Section 6.4, "Oracle Java ME SDK Directories" and Table 6–1. Launch the demo. The application displays the contents of `root1`. Select your SVG file and choose the Open soft key.

22.1.2 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 dynamically sized to exactly fit the display area. The output is clear and sharp.

22.1.3 Play SVG Animation

This application plays an SVG animation depicting a Halloween greeting card. Press 8 to pause, 5 to start or resume, and 0 to stop.

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();
```

22.1.4 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.

22.1.5 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.

22.1.6 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 adapts 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 provided both the artist and the developer observe the SVG document structure conventions.

22.1.7 Picture Decorator

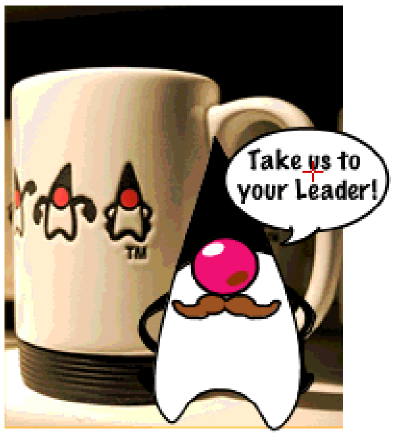
In this sample you use the phone keys to add decorations to a photograph. The key values are:

Key	Action
1	key shrink
2	key next picture
3	key grow
4	key help
5	key horizontal flip
6	key vertical flip
7	key rotate counter-clockwise
8	key previous picture
9	key rotate clockwise
#	display picker options

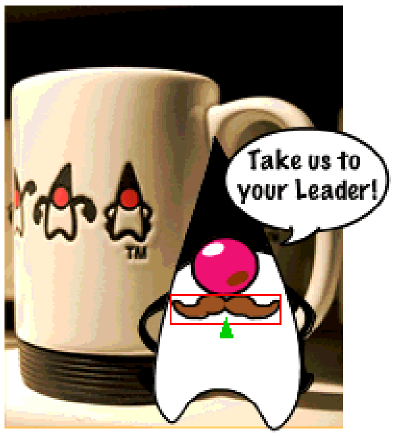
This demo provides 16 pictures for you to decorate.

Use the 2 and 8 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, indicating it is selected and can be moved, resized, and manipulated.



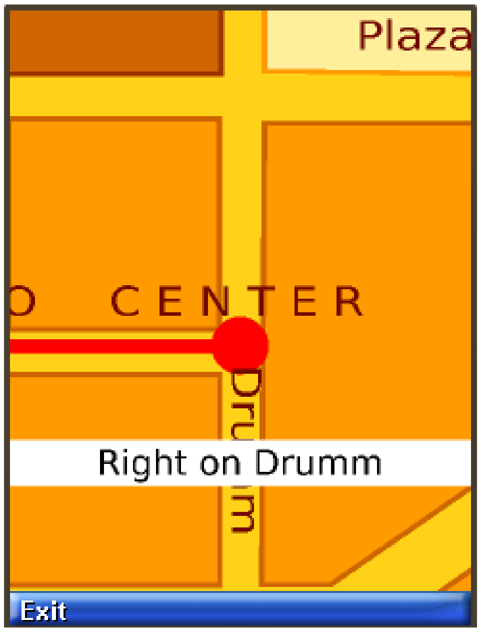
Use the navigation arrows to move the graphic. Use 1 to shrink the graphic, and 3 to enlarge the graphic. Use 5 or 6 to flip, and 7 or 9 to rotate. When you are satisfied with the position, press Select. Look for a green triangle. This is a cursor. Use the navigation keys to move the green triangle around 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 and it can be manipulated or removed.



To remove a decoration (a property), select an object, then click the Menu soft key and choose Remove prop.

22.1.8 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.



22.2 Running SVGContactList

This application uses different skins to display the same contact list information and a news banner. The skins feature 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 selected name is marked with a special character (a > or a dot) and is displayed in a larger font.

Press > or the select button to see more information for the selected name. When you are in the detailed view you can traverse the detail entries using the up or down arrows.

Press < or the select button to return to the contact list.

Press the left soft button to go back to the demos MIDlet list and view another skin.

23 JSR 239: Java Bindings for Open GL ES

JSR 239 provides a Java language interface to the open standard OpenGL ES graphics API.

23.1 Open GL 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.

24 JSR 256: Mobile Sensor API Support

The JSR 256 Mobile Sensor API allows Java ME application developers to fetch data from sensors. A sensor is any measurement data source. Sensors can vary from physical sensors such as magnetometers and accelerometers to virtual sensors that combine and manipulate the data they have received from various kinds of physical sensors. An example of a virtual sensor might be a level sensor indicating the remaining charge in a battery or a field intensity sensor that measures the reception level of the mobile network signal in a mobile phone.

JSR 256 supports many different types of sensor connection (wired, wireless, embedded and more) but this SDK release only provides preconfigured support for sensors embedded in the device.

The SDK GUI provides sensor simulation. The emulator's External Events Generator Sensors tab enables you to run a script that simulates sensor data.

You can use the API available with the SDK to create a custom sensor implementation with additional capabilities and support for different connection types.

The Sensors demonstration has two MIDlets, SensorBrowser and Marbles that demonstrate the SDK's implementation of the Mobile Sensor API.

24.1 Creating a Mobile Sensor Project

Follow these steps to create a mobile sensor project:

1. Select File > New > MIDlet Project and fill in the information in the Create a MIDlet Project dialog box. Click Next.
2. On the MIDlet Project Content page, change the Microedition Configuration version to 1.1. Ensure that the MIDP version is 2.1. Click Finish
3. Expand the finished project in the Package pane and double-click Application Descriptor to open the Overview pane.
4. Click the Application Descriptor tab at the bottom of the Overview pane and add configuration values for the embedded sensors shipped with the ME SDK as described below.

A sensor project freely detects sensors, but this does not imply you can get data from the sensors you find. You might need to explicitly set permissions in your project so you can interact with certain sensors. To see an example, open the Sensors sample project. Right-click on Samples and select Properties, choose the Application Descriptor category, and select the API Permissions tab.

The following permissions work with the preconfigured embedded sensors shipped with the SDK:

- `javax.microedition.io.Connector.sensor`

  Required to open a sensor connection and start measuring data.
- `javax.microedition.sensor.ProtectedSensor`

  Required to access a protected sensor.
- `javax.microedition.sensor.PrivateSensor`

  Required to access a private sensor.

A sensor is private or protected if the sensor's security property has the value private or protected. The security property is an example of a sensor property you might create for yourself in your own sensor configuration. You can create your own optional properties using `com.sun.javame.sensor`*N*`.proplist` and `com.sun.javame.sensor`*N*`.prop.`*any_name*, where *N* is the sensor number and *any_name* is the name of your property. The security sensor property was created as follows:

```
# add security into proplist
com.sun.javame.sensor<N>.proplist: security
# add security property value
com.sun.javame.sensor<N>.prop.security: private
```

24.2 Using a Mobile Sensor Project

The sample Sensor project can be installed over the air. To install the application into the emulator right-click on Samples and select Properties, choose the Running category, select Execute through OTA, and click OK.

In the emulator window, select Device>Sensors. In this tabbed pane, you can view all sensors that are currently available in the emulator (sensor ID, name, and availability are listed). If the sensor supports change to availability you can click on the check box to change it. As mentioned earlier, the provided implementation does not support availability change, but a custom implementation you create might do so.

When you select a sensor row, the bottom of the dialog displays any custom sensor controls. For example, the acceleration sensor, has three channels: axis_x, axis_y, and axis_z. Each channel has a slider that changes the current channel value, and an edit box you can use to input a value. The channel unit label is displayed on the far right.

Under the channels there is a script player control that enables you to play sensor value events from a script file of the format discussed in Section 24.3, "Creating a Sensor Script File." See Section 24.4, "SensorBrowser" for a description of how to use the Sensors demo.

24.3 Creating a Sensor Script File

To simulate sensor inputs, provide a sensor script. The file format is as follows:

```
<sensors>
   <value time="0">
      <channel id="0" value="0" />
      <channel id="1" value="0" />
   </value>
   <value time="100">
      <sensor active="false"/>
   </value>
   <value time="100">
      <channel id="0" value="-50" />
      <channel id="1" value="10" />
     <sensor active="true"/>
   </value>
</sensors>
```

marbles.xml in the Sensors project directory is an example of a sensor script file. The attributes are as follows:

- The attribute time in the value tag is the delay from the previous command in milliseconds.
- The `channel` tag sets the value of the channel with the specified id value, to value. The channel ignores the id if the value of id is not specified or if the value is out of the channel range.
- The `sensor` tag is a true or false value that makes the sensor available or unavailable. The preconfigured sensors provided with this release are embedded, so they cannot be deactivated. If you configure your own sensor that is not embedded, it is possible to deactivate it.

24.4 SensorBrowser

The SensorBrowser application displays the sensor detail information for reach channel defined for the demo.

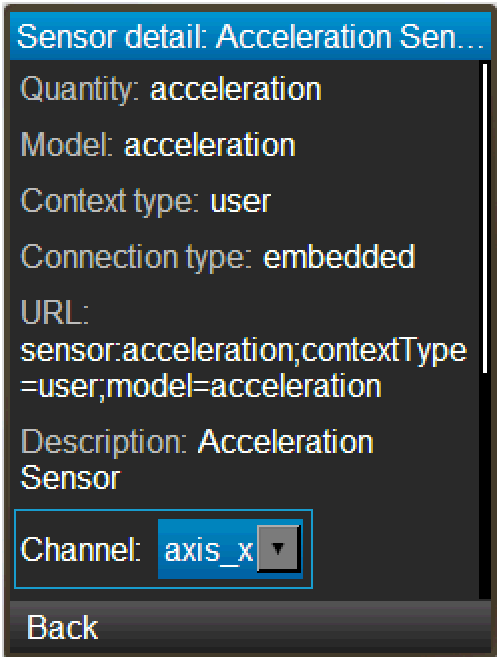
1. In the emulator select SensorBrowser and use the soft key to select Launch the application.

   Depending on your security settings you might see the warning: "Sensors" wants to connect to sensor <#>. Is it OK to use sensor? For test purposes, select "Ask once per application use" and choose the Yes soft button.

   The emulator displays a list of sensors.

2. Use the navigation keys to highlight a sensor, then use the soft key to select Detail.

   For example, the following screen shows the details for the acceleration sensor.


Sensor detail: Acceleration Sen...

Quantity: acceleration

Model: acceleration

Context type: user

Connection type: embedded

URL:
sensor:acceleration;contextType
=user;model=acceleration

Description: Acceleration
Sensor

Channel: axis_x

Click Back, then click Exit to return to the application menu.

24.5 Marbles

This demonstration uses the Marbles game to provide visual feedback for sensor inputs provided in a script.

1. From the application menu select Marbles and use the soft key to launch the application.
2. In the emulator, select Device > Sensors to open the external events generator.

   The emulator displays a list of the sensors in this application.
3. Select the Acceleration Sensor row (ID 3).
4. Click the Browse button, and in the Sensors project directory choose *marbles.xml*.
5. Observe the movement of the marbles on the emulator screen. On the external events screen you can see the sliders move as the script runs. You can use the familiar controls to play, pause, and stop the script.

25 JSR 257: Contactless Communication API

The Contactless Communication API (http://jcp.org/en/jsr/detail?id=257) is a Java ME optional package that allows applications to access information on contactless targets, such as Radio Frequency Identification (RFID) tags and bar codes. RFID tags are often used in business for item identification, article surveillance, and inventory. Each RFID tag contains a unique identification number used to identify a tagged object.

Using the JSR 257 API, an RFID reader can be built into an Oracle Java Wireless Client software phone stack, allowing the handset to read data from a tagged target and write data back to it. RFID readers use the 13.56 MHz radio frequency and the communication distance is usually less than 10 centimeters.

The Near Field Communication (NFC) Forum defines the NFC Data Exchange Format (NDEF) data packaging format. NDEF facilitates communication with an RFID tag, or between one NFC device and another. The Contactless Communication API provides a connection to any physical target that supports the NDEF standard, allowing applications to exchange data with any target tagged with NDEF formatting, regardless of actual physical type.

For an explanation of this implementation, see the *Oracle Java Wireless Client Porting Guide*.

25.1 Using ContactlessDemo

The Oracle Java ME SDK provides a way to test contactless communication. The MIDlet running on the emulator waits to detect an RFID tag. You can simulate the tag communication using the emulator's external events generator to detect and attach the tag. You can use one of the tags included in the sample, or create tag files of your own, as described in Section 25.2, "Tag File Formats."

1. Launch the ContactlessDemo. The MIDlet registers the RFID tag listener, the NDEF tag listener and the NDEF record listener, then notifies you that it is waiting for a tag.
2. In the emulator, choose Contactless Communication. In the external events generator the tag emulator supplies several tags by default: hello, nested, vcard, jdts, jdts2, and ndefEmpty.
3. To test the connection, select an available tag and press the Attach tag button.

   In the emulator the MIDlet notifies you that the NDEF target is detected, displays the tag information, and prints the payload if it is a text record.

   In the external events generator, press the Detach tag button to end the session.

Events are recorded in the log area. To clear the log, right-click and select delete text. To clear the emulator screen press the Clear soft button.

4. To create your own tag, create a tag file according to the NDEF standard. For a sample, see Section 25.2, "Tag File Formats."

   In the external events generator, press the Create tag button, browse to select your tag file, and press Open. If the file is properly formed, the new tag is added to the available tags list.

   You can use the Remove tag button to remove any tag from the list. If it's a tag you created, the original file on disk is not affected. If the default tags are removed, they reappear when you restart the demo.

5. Optional. Instead of performing interactive actions in the external events generator, you can use a script to do the same thing.

   Create a file as directed in Section 25.3, "Script Format." In the external events generator, click the Browse button to locate your script, then press Play.

25.2 Tag File Formats

Tags are created in XML format in accordance with the NFC and NDEF standards. To see how the sample files are formed, see: *installdir*`\toolkit-lib\modules\emulator-ui-window-external-events\jsr257\conf\tags`.

A sample file with several records might look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<jsr257client>
    <UID>12-CD-45-67-89-AB-CD</UID>

    <TargetProperties>
        <TargetProperty>NDEF</TargetProperty>
    </TargetProperties>

    <NDEFMessage>
        <NDEFRecord>
            <Format>MIME</Format>
            <Name>text/plain</Name>
            <Id>mimeid</Id>
            <Payload>Hello, MIME World!</Payload>
        </NDEFRecord>
        <NDEFRecord>
            <Format>MIME</Format>
            <Name>text/example</Name>
            <Id>urn:company:product:ndef:payload:2</Id>
            <Payload>payload2</Payload>
        </NDEFRecord>
        <NDEFRecord>
            <Format>EXTERNAL_RTD</Format>
            <Name>urn:nfc:ext:oracle.com:type1</Name>
            <Id></Id>
            <Payload>payload3</Payload>
        </NDEFRecord>
        <NDEFRecord>
            <Format>URI</Format>
            <Name>urn:company:product:test_uri</Name>
            <Id>urn:company:product:ndef:payload:4</Id>
            <Payload>payload4/<Payload>
        </NDEFRecord>
        <NDEFRecord>
            <Format>NFC_FORUM_RTD</Format>
            <Name>urn:nfc:wkt:Sp</Name>
            <Id></Id>
            <Payload>smart-poster</Payload>
        </NDEFRecord>
        <NDEFRecord>
            <Format>MIME</Format>
            <Name>text/x-vCard</Name>
            <Id>duke</Id>
            <Payload>BEGIN:VCARD VERSION:2.1 FN:Oracle TEL:+1-650-506-7000
                     ADR:500 Oracle Parkway City:Redwood Shores
                     State:CA;94065 END:VCARD
            </Payload>
        </NDEFRecord>
    </NDEFMessage>
</jsr257client>
```

25.3 Script Format

You can use the external events generator buttons to attach and detach a tag, or you can write a script to perform these actions. The script syntax is as follows:

```
# Comment:
  # this is a comment
# Tag definition:
    tag <tag name> <path to the tag xml file>
# Attach tag:
   attach <tag name>
# Delay. Ensures the tag is attached before other actions.
   wait <time in ms>
# Print tag information:
   print <tag name>
# Detach tag:
   detach <tag name>
```

This is a sample script:

```
   tag C D:\MyTags\ccomtag.xml
   attach C
   print C
   wait 10000
   detach C
```

In the external events generator click Browse and choose the script file, then press Play to run the script. The results are shown in the Log area. For example, if the sample script calls the sample tag file in Section 25.2, the log output is as follows:

```
[18:24:10]  Run Script: D:\JMESDKLocal\ccomtag.xml
[18:24:10]  Define tag 2058 (C)
[18:24:10]  Print 2058 (C)
[18:24:10]  Attached tag 2058 (C)
[18:24:10]  UID: 02-34-56-78-9A-BC-DE
Properties:  NDEF
NDEF message: 8 record(s)
#0: NDEF record:  format=MIME, name=text/plain, id.length=2, payload.length=18
payload=Hello, MIME world!
#1: NDEF record:  format=MIME, name=text/example, id.length=34, payload.length=8
```

```
payload=payload2
#2: NDEF record:  format=EXTERNAL_RTD, name=oracle.com:type1, payload.length=8
payload=payload3
#3: NDEF record:  format=URI, name=urn:company:product:test_uri, id.length=34,
payload.length=8
payload=payload4
#4: NDEF record:  format=EXTERNAL_RTD, name=company.com:type1, id.length=34,
payload.length=8
payload=payload5
#5: NDEF record:  format=NFC_FORUM_RTD, name=Sp, payload.length=12
payload=smart-poster
#6: NDEF record:  format=URI, name=message/http, id.length=3, payload.length=56
payload=http://www.oracle.com/technetwork/java/javame/index.html
#7: NDEF record:  format=MIME, name=text/x-vCard, id.length=4, payload.length=122
payload=BEGIN:VCARD VERSION:2.1 FN:Oracle TEL:+1-650-506-7000
ADR:500 Oracle Parkway; City:Redwood Shores;State:CA;94065 END:VCARD
[18:24:10]  Wait 10000ms
[18:24:20]  Detached tag 2058 (C)
[18:24:20]  Script finished.
[18:24:25]  Received data for unknown tag 2,058
```