Oracle® Java Micro Edition Software Development Kit

Developer's Guide

Release 3.2 for Windows

E24265-04

September 2012

This document describes how to use the Java ME SDK plugin for NetBeans.

ORACLE®

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

E24265-04

Copyright © 2009, 2012, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

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:

- The supported API documentation links can be found in Table 16–1, " Supported JCP APIs".

- 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), Information Module Profile - Next Generation (IMP-NG), and Connected Device Configuration (CDC) platforms.

- Section 1.1, "Quick Start"
- Section 1.2, "Tips for Legacy Toolkit Users"
- Section 1.3, "Java ME SDK Update Center"

As of version 3.2, the Java ME SDK is a plugin to the NetBeans IDE. In NetBeans the Mobility Pack is a prerequisite for installing the Java ME SDK.

The Java ME SDK is also a plugin to the Eclipse IDE. This documentation does not discuss the Eclipse IDE. A separate online help system supports the Oracle Java ME SDK for Eclipse.

1.1 Quick Start

The Oracle Java ME SDK plugin uses NetBeans technology, as described in the NetBeans 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 JavaHelp Online Help viewer. Remember to use the search capability and the index 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.

- See the Projects window and the Files 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. See Section 4.3, "Working With Projects".
- A project has a default device that is used when you run it from the toolbar (the green arrow), Run > Run Project, or Run on the project's context menu. To see a project's default device, right-click the project and select Properties. Choose the Platform category and you see the default device displayed in the Device field. To reset the Device make another choice from the dropdown menu.

  To run an application on different devices without changing the default device, right-click on the project and select Run With from the context menu. Choose a different device and click OK.
- The emulator is an independent process, and when it has started it is a separate process from the build process running in NetBeans. 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".

  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. If you want to create your own device, see Section 6.11, "Using the Custom Device Skin Creator".

1.2 Tips for Legacy Toolkit Users

If you previously used the Sun Java Wireless Toolkit for CLDC or the CDC Toolkit, the advice in Section 1.1, "Quick Start" still applies. Although the user interface is quite different, the project concept is similar. The following tips apply legacy terms and ideas to the SDK.

- Runtime focus is less on the project and more on device capabilities and the emulation process.

  In legacy toolkits you had to be careful to match the platforms, the APIs, and the capability of the output device. The SDK handles this problem differently, but as described in Section 2.2.1, "Java ME Platforms", you should be sure that the emulator platform is correct and a device profile is selected.

  As mentioned in the Section 1.1, "Quick Start", clicking the green arrow runs the main project (to set the main project select Run > Set Main Project and select a project from the dropdown menu). Alternatively, you can right-click any open project and select run.

  In the device selector (Tools > Java ME > Device Selector) you can test many devices without changing the project properties. Right-click any device and choose Run. Only projects that are compatible with the device are shown in the context menu.
- Import applications from legacy toolkits to SDK projects. The installation of the legacy toolkit must exist on the host machine. See Section 4.2.4, "Import a Legacy MIDP Project", Section 2.2.2, "Create a Platform for Legacy CDC Projects", and Section 4.2.5, "Import a Legacy CDC Project".

- Legacy toolkit **settings** are Application Descriptors in the SDK. Right-click on a project and select Properties. Choose the Application Descriptor category.
- Legacy toolkit **utilities** are generally accessible from Tools > Java ME submenu in the NetBeans IDE. For example, the WMA console, the Java ME SDK Update Center and more can be started from the Tools > Java ME submenu.

  For example, select Tools > Java ME > WMA Console in the NetBeans IDE to see the WMA Console output.
- Profiling output and Network monitoring utilities are accessed from the Profile > Java ME submenu in the NetBeans IDE.
- The emulator is familiar, but there are some fundamental differences.

  It is important to realize that the emulator is a remote process, and when it starts it is independent of the build process running in NetBeans. 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 from the emulator. For more on this topic, see Section 3.2, "Running a Project" and Section 4.3, "Working With Projects".

  In the Wireless Toolkit you could simultaneously run multiple versions of a device because the toolkit would increment the phone number automatically each time you launched a project. Because the emulator is now a remote process, the phone number is a unique property that must be set explicitly for the device instance.

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

  The emulator has additional display functionality. See Section 6.9, "Emulator Features".

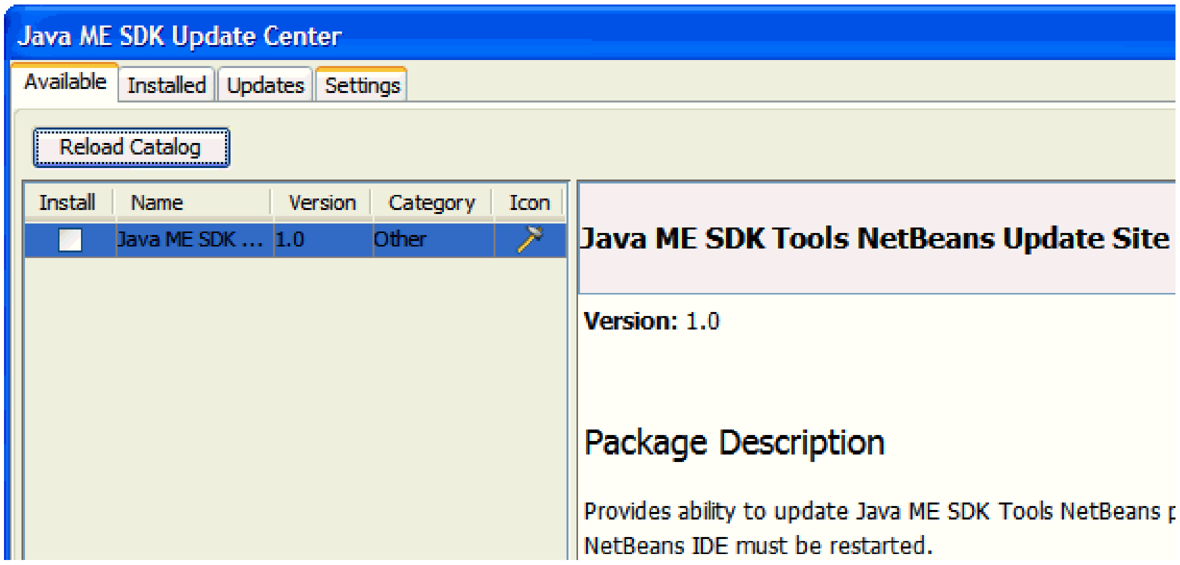
1.3 Java ME SDK Update Center

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 Tools > Java ME > Java ME SDK Update Center. The update center uses the same technology as the NetBeans Plugins Manager. The update manager works separately from NetBeans so that the Java ME SDK plugin can be updated independently.

Java ME SDK is delivered as three NetBeans plugins in their own category named Java ME SDK Tools. The plugins are:

- Java ME SDK Tools
- LWUIT Resource Editor
- Java ME SDK Demos

To detect updates, select Tools > Java ME > 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.



Install	Name	Version	Category	Icon
☐	Java ME SDK ...	1.0	Other	

Java ME SDK Tools NetBeans Update Site

Version: 1.0

Package Description

Provides ability to update Java ME SDK Tools NetBeans p
NetBeans IDE must be restarted.

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, please 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.

2 Platforms

This chapter describes the Oracle Java ME SDK technology platforms, also called stacks. They are: CLDC with MIDP, IMP-NG and CDC, 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
- CDC with Advanced Graphics and User Interface (AGUI)
- CDC with Personal Basis Profile (PBP)

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

See Section 2.1.1, "CLDC with MIDP", Section 2.1.2, "IMP-NG" and Section 2.1.3, "CDC".

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 16.1, "JCP APIs".

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 3.7, "Running MIDP and CLDC Sample Projects", Section 4.2.1, "Create a CLDC Project", Chapter 6, "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 as 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 16.2, "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 3.8, "Running IMP-NG Sample Projects".

2.1.3 CDC

A Java ME Platform, Connected Device Configuration (CDC) (`http://jcp.org/en/jsr/detail?id=218`) application is an application targeted for network-connected consumer and embedded devices, including high-end mobile phones, smart communicators, high-end PDAs, and set-top boxes.

Devices that support CDC typically include a 32-bit microprocessor or controller and make about 2 MB of RAM and 2.5 MB of ROM available to the Java application environment.

CDC is based upon the open source project phoneME Advanced, found at `http://java.net/projects/phoneme`. A CDC application conforms to the Connected Device Configuration with a set of profiles that include Personal Basis Profile, Foundation Profile, and AGUI:

- CDC 1.1 with PBP 1.1 (`http://jcp.org/en/jsr/detail?id=217`)
- Foundation Profile 1.1 (`http://jcp.org/aboutJava/communityprocess/final/jsr219/index.html`)
- AGUI 1.0 (`http://www.jcp.org/en/jsr/detail?id=209`)

The SDK provides three default emulators to support CDC:

- Default CdcPbpPhone1

  CDC 1.1, PBP 1.1, Foundation Profile (FP) 1.1
- VgaAGUIPhone1

  CDC 1.1, PBP 1.1, FP 1.1 and AGUI 1.0
- VgaCdcPhone1

  CDC 1.1, PBP 1.1, FP 1.1

See Table 3–2, " CDC Sample Projects", Section 4.1, "Project Types" and Section 6.3, "Viewing Device Properties".

2.2 Managing Java Platforms

To view the Java Platform Manager, select Tools > Java Platforms. Alternatively, right-click on a project, choose Properties from the context menu, select the Platform category, and select the Manage Emulators button to open the Java Platform Manager.

The Java Platform Manager is a tool for managing different versions of the Java Development Kit (JDK) and customizing Java platforms that your applications depend on. You can add source files and Javadoc documents to the existing platforms.

The Oracle Java ME SDK pre-registers CDC, CLDC/MIDP and Java SE (the JDK serves as the default platform) for version 3.2. These platforms have similar options:

Devices. (CLDC) View all the CLDC and IMP-NG devices that the Device Manager has discovered. Click Refresh to reconfigure the platform and refresh the list.

Classes. (CDC, Java SE) View the platform's classpaths. Add a JAR or folder containing additional classes. A class’s location in the list determines its place in the classpath. Use the Move Up and Move Down buttons to change the class position.

Sources. Add JAR files or source files to the Sources tab to register source code.

Javadoc. Add Javadoc documentation to support any new classes or source files you have added.

See Section 2.2.1, "Java ME Platforms".

2.2.1 Java ME Platforms

In the Oracle Java ME SDK the platforms are embedded Java runtimes specifically for resource-constrained devices. Because the Netbeans Mobility pack is installed for Java ME you see the legacy 3.0.5 platforms coexisting with version 3.2 platforms and devices. Each platform has its own set of devices and optional packages.

> **Note:** If you cannot see the 3.2 devices in the Device Selector, choose Tools > Java ME > Active Device Manager and select Java(TM) ME Platform SDK 3.2.

Applications that worked in previous platform versions might not run on the current version, and vice versa. Follow these steps to set a project's platform options.

1. Right-click on a project and choose Properties from the context menu.
2. Select the Platform category. Be sure that the Emulator Platform is set to version 3.2.
3. The device configuration should be automatically selected, but the Device Profile might not be explicitly selected. Be sure to choose a profile.
4. Optional. Check any optional packages that are required to support the current project. (If this is an IMP-NG project you also see the Oracle APIs listed as optional packages. See Section 16.2, "Oracle APIs").

   Click OK.
5. Rebuild the project and run.

2.2.2 Create a Platform for Legacy CDC Projects

The Oracle Java ME SDK version 3.2 platform name for CDC does not match the name in the legacy CDC toolkit and the CDC Mobility Pack. The legacy name is "Sun Java Toolkit 1.0 for Connected Device Configuration" while the SDK name is "CDC Oracle Java(TM) Platform Micro Edition SDK 3.2". To ensure a successful import, you can create a new platform and give it the legacy name.

The following procedure enables you to import legacy CDC projects without Reference errors (see Section 3.6, "Resolving Reference Problems").

1. Select Tools > Java Platforms. Select "CDC Oracle Java(TM) Platform Micro Edition SDK 3.2", and in the Classes tab, note the libraries required for the platform.
2. Click Add Platform... and click Next.
3. Select Java ME CDC Platform Emulator and click Next.
4. On the Choose Platform page, select the SDK installation directory. Click Next.
5. On the Platform Name page, type "Sun Java Toolkit 1.0 for Connected Device Configuration" in the Name field. In the Sources tab, add the following libraries: `agui.jar`, `cdc_1.1.jar`, `fp_1.1.jar`, `pbp_1.1.jar`, and `secop_1.0.jar`.

   Click Finish, and Close.

See Section 4.2.5, "Import a Legacy CDC Project" and Section 3.6, "Resolving Reference Problems".

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 1.3, "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, "Resolving Reference Problems"
- Section 3.7, "Running MIDP and CLDC Sample Projects"
- Section 3.8, "Running IMP-NG Sample Projects"

3.1 Creating a Sample Project

Sample applications are installed in a separate NetBeans 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 NetBeans 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:

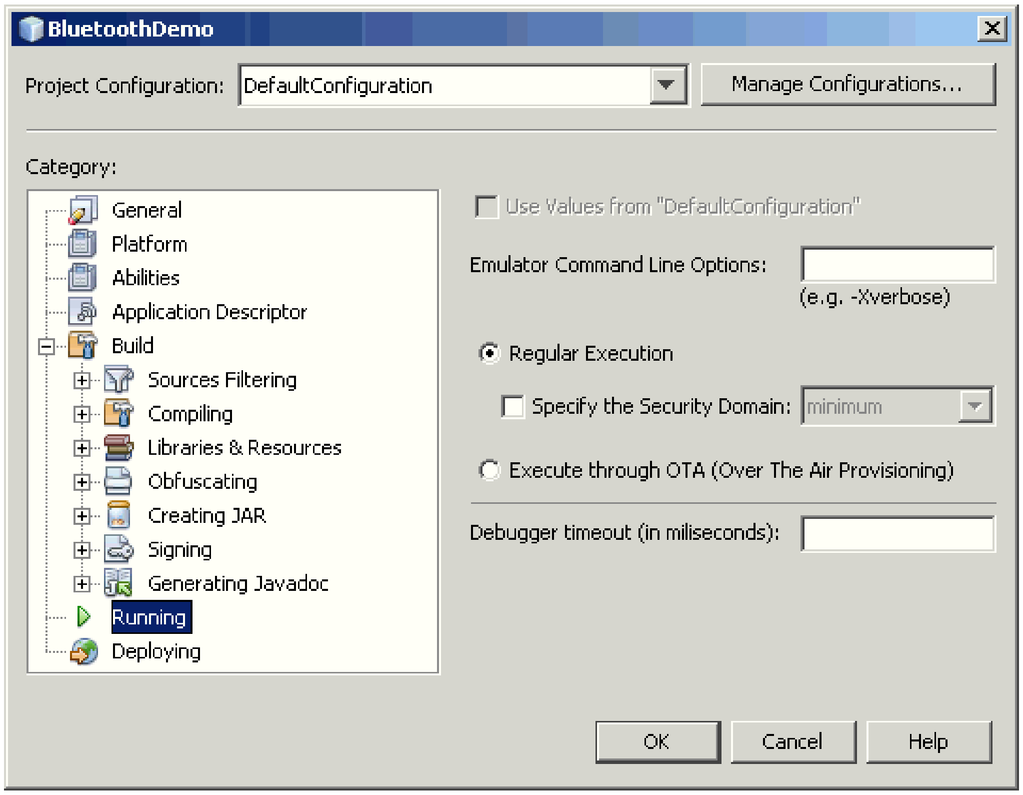
userhome`\My Documents\NetBeansProjects\WMADemo\src\example\sms\SMSSend.java`

1. Go to File > New Project and in the Categories window select Samples > Java ME SDK 3.2 and single-click a sample project name. Click Next.
2. Accept Name and Location page default values or provide your own. Click Next.
3. Choose an emulator platform and a device. Note, changing the device affects the possible device profiles. Click Finish.

   The project is added to the Project window.

   > **Note:** If you can't see the Project window choose Window > Projects. To see console output, select Window > Output > Output.

4. Set the project's default execution mode. Right-click on a project and select Properties from the context menu, then choose the Running category.



Project Configuration: DefaultConfiguration

[ ] Use Values from "DefaultConfiguration"

Emulator Command Line Options: (e.g. -Xverbose)

[✓ Regular Execution]

[ ] Specify the Security Domain: minimum

Execute through OTA (Over The Air Provisioning)

Debugger timeout (in miliseconds):

   If you want to install the application in the emulator each time it is run, select Execute through OTA (Over the Air Provisioning). This mode is required if your application uses the push registry or the external events generator, and optional for other cases.

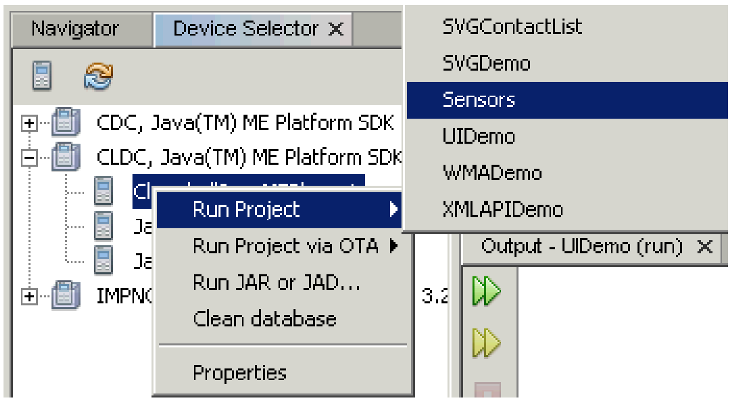
   If the application does not need to be installed in the emulator, choose Regular execution (see Section 13.1, "Security Domains").

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.

- Right-click the project and select Run from the context menu.
- To run the main project (which is shown in bold text in the Projects window), click the green Run button in the toolbar or press F6. To set the main project, select Run > Set Main Project and select a project from the dropdown menu.
- To run the project on a different device, or to change the execution mode, choose the device in the Device Selector window (Tools >Java ME > Device Selector). Right-click on a device and select Run Project or Run Project via OTA from the context menu. Pull right to see a listing of open projects. Projects that cannot run on the current device are grayed out.



- Choose Start > Programs > Java(TM) ME Platform SDK 3.2 > Java ME SDK CLDC Emulator.

  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.

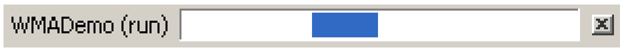
2. As the sample project runs, soft keys might be enabled below the screen on the left or right side.

   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 or Table 3–2.

3. When you are finished viewing the 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. Pressing the red stop button in the NetBeans Output window terminates the build script, but it does not close the emulator instance. You can also terminate the build script by clicking the X next to the progress meter at the bottom of the IDE.



   Likewise, closing the NetBeans 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.

   This ensures 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 14.3, "Emulator Command Line Options".

  MIDlet Suites use `runMIDlet` to perform the installation.

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

  CDC platforms install applications as follows:

  installdir`\runtimes\cdc-hi\bin\cvm`.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` and `cvm` 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 3.7.1, "Running the AdvancedMultimediaSupplements Sample Project"
AudioDemo	MMAPI 1.1	Demonstrates audio capabilities, including mixing and playing audio with an animation.	Section 19.4, "Running AudioDemo"
BluetoothDemo	JSR 82	Demonstrates device discovery and data exchange using Bluetooth.	Section 18.2, "Running the Bluetooth Demo"
CHAPIDemo	JSR 211	A content viewer that also uses MediaHandler.	Section 25.4, "Running the CHAPIDemo Content Browser"
CityGuide	JSR 179	A city map that displays landmarks based on the current location.	Section 22.2, "Running the CityGuide Sample Project"
ContactlessDemo	JSR 257	Emulates detection of RFID tags.	Section 29.1, "Using ContactlessDemo"

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

Sample	Optional Package	Description	Instructions
Demo3D	JSR 184	Contains MIDlets that demonstrate how to use 3D graphics, both immediate mode and retained mode.	Section 24.4, "Running Demo3D Samples"
Demos	MIDP 2.0	Includes various examples: animation, color, networking, finance, and others.	Section 3.7.2, "Running the Demos Sample Project"
FPDemo	CLDC 1.1	Simple floating point calculator.	Section 3.7.3, "Running FPDemo"
Games	MIDP 2.0	Includes TilePuzzle, WormGame, and PushPuzzle.	Section 3.7.4, "Running Games".
JSR172Demo	JSR 172	Demonstrates how to use the JSR 172 API to connect to a web service from a MIDlet.	Section 20.3, "Run JSR172Demo"
LWUITBrowser	N/A	Demonstrates LWUIT features.	Chapter 12, "Lightweight UI Toolkit"
LWUITDemo	N/A	Demonstrates LWUIT features.	Chapter 12, "Lightweight UI Toolkit"
LWUITIODemo	N/A	Demonstrates LWUIT features.	Chapter 12, "Lightweight UI Toolkit"
LWUITMakeover	N/A	Demonstrates LWUIT features.	Chapter 12, "Lightweight UI Toolkit"
LWUITSpeed	N/A	Demonstrates LWUIT features.	Chapter 12, "Lightweight UI Toolkit"
LWUITTimeZone	N/A	Demonstrates LWUIT features.	Chapter 12, "Lightweight UI Toolkit"
LWUITTipster	N/A	Demonstrates LWUIT features.	Chapter 12, "Lightweight UI Toolkit"
MMAPIDemos	MMAPI	Demonstrates MMAPI features, including tone sequences, MIDI playback, sampled audio playback, and video.	Section 19.5, "Running MMAPIDemos"
Multimedia	MMAPI	Demonstrates different video playback formats.	Section 19.5.3, "Video"
NetworkDemo	MIDP 2.0	Demonstrates how to use datagrams and serial connections.	Section 3.7.5.1, "Socket Demo" and Section 3.7.5.2, "Datagram Demo"
ObexDemo	JSR 82	Demonstrates device discovery and data exchange using Bluetooth.	Section 18.3, "Running the OBEX Demo"
PDAPDemo	JSR 75	Demonstrates how to manipulate contacts, calendar items, and to-do items. Demonstrates accessing local files.	Section 17.3, "Running PDAPDemo"
PhotoAlbum	MIDP 2.0	Demonstrates a variety of image formats.	Section 3.7.6, "Running PhotoAlbum"
SATSADemos	JSR 177	Demonstrates communication with a smart card and other features of SATSA.	Section 21.4, "Running SATSADemos"
SATSAJCRMIDemo	JSR 177	Shows how to use the SATSA-Java Card Remote Invocation method.	Section 21.4.5, "Running SATSAJCRMIDemo"
Sensors	JSR 256	The SensorBrowser and Marbles game demonstrate sensor input.	Section 28.4, "SensorBrowser" and Section 28.5, "Marbles"

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

Sample	Optional Package	Description	Instructions
SVGContactList	JSR 226	Uses SVG to create a contact list displayed with different skins.	Section 26.2, "Running SVGContactList"
SVGDemo	JSR 226	Uses different SVG rendering techniques.	Section 26.1, "Running SVGDemo"
UIDemo	MIDP 2.0	Showcases the breadth of MIDP 2.0's user interface capabilities.	Section 3.7.7, "Running UIDemo"
WMADemo	WMA 2.0	Shows how to send and receive SMS, CBS, and MMS messages.	Section 23.2, "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.

Table 3–2 lists the CDC sample projects available in this release.

Table 3–2 CDC Sample Projects

Sample	Optional Package	Description	Instructions
AGUIJava2DDemo	JSR 209	This stand-alone application is a Java SE application adapted for the CDC environment. It demonstrates the graphical and animation capabilities of the Java 2D™ API.	Click the blue arrows to page through the various images and animations. The applications focus on curves. Click the AA icon to see how antialiasing affects appearance.
AGUISwingSet2	JSR 209	Functional tools such as buttons, sliders, and menus implemented with Swing.	Click through the tabs to view the controls and animations.

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.

Note: CDC emulators do not work through a proxy. Communications such as downloading images from the Internet fail on CDC emulators.

The settings are typically the same as those you are using in your web browser.

1. Select Tools > Options > General.
2. Choose a Proxy Setting:
   - No Proxy
   - Use System Proxy Settings
   - Manual Proxy Settings

   To set the HTTP Proxy, fill in the proxy server address field and the port number. The HTTP Proxy Host is the host name or numeric IP address of the proxy server used to connect to HTTP and FTP sites. The Proxy Port is the port number of the proxy server.

To set the HTTPS or Socks proxy, click More and fill out the Advanced Proxy Options form.

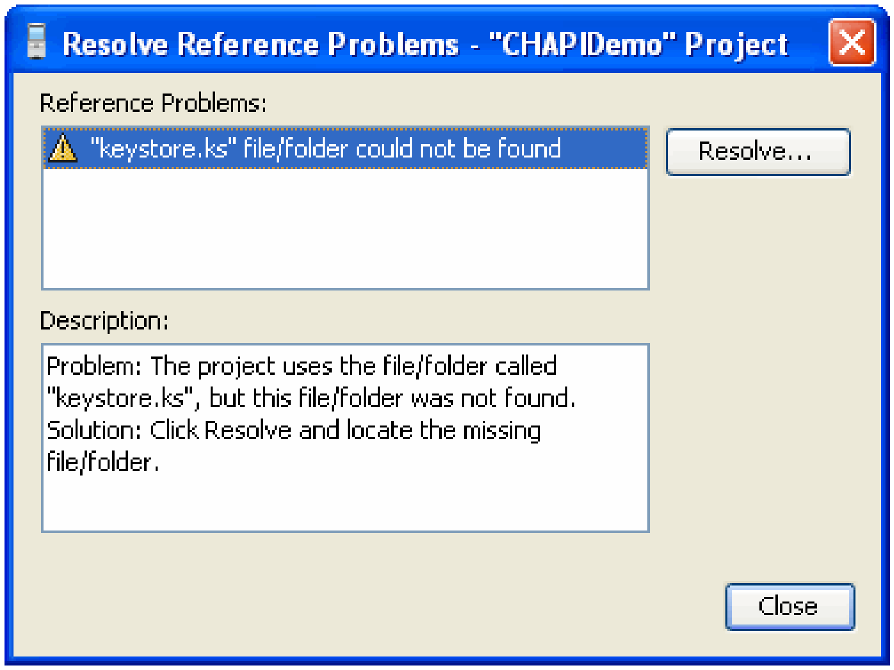
3.6 Resolving Reference Problems

Sometimes when you open a project you can see it has a reference warning. In the Projects tab the project name is red, and the icon shows a warning symbol, as seen below:

Usually this warning means the project refers to a platform, file, or library that cannot be found.

If you are using an old project it might be referring to a platform or device that is not installed. In this case edit the project properties and select an available platform and device as described in Section 2.2.1, "Java ME Platforms", then rebuild the project.

If the problem is not the platform, right-click on the project and choose Resolve Reference Problems.



"keystore.ks" file/folder could not be found

Problem: The project uses the file/folder called "keystore.ks", but this file/folder was not found.
Solution: Click Resolve and locate the missing file/folder.

The window displays the missing file, the problem, and a possible solution. In this case the project probably used a literal path to the file `keystore.ks`. Clicking the Resolve... button opens a file browser so you can find the missing keystore file. The default location is as follows:

installdir`\runtimes\cldc-hi\lib`

Locate and select the file. You receive confirmation that the problem is resolved, and you can now click Close.

3.7 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 3.7.1, "Running the AdvancedMultimediaSupplements Sample Project"
- Section 3.7.2, "Running the Demos Sample Project"
- Section 3.7.3, "Running FPDemo"
- Section 3.7.4, "Running Games"
- Section 3.7.5, "Running Network Demo"
- Section 3.7.6, "Running PhotoAlbum"
- Section 3.7.7, "Running UIDemo"

For other CLDC demos, see Table 3–1.

3.7.1 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 3.7.1.1, "Image Effects"
Section 3.7.1.3, "Camera"
Section 3.7.1.4, "Moving Helicopter"
Section 3.7.1.2, "Music Effects"

3.7.1.1 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.

3.7.1.2 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.

3.7.1.3 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.
- 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`

3.7.1.4 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.

3.7.2 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 3.7.2.1, "Colors"
- Section 3.7.2.2, "Properties"
- Section 3.7.2.3, "Http"

- Section 3.7.2.4, "FontTestlet"
- Section 3.7.2.5, "Stock"
- Section 3.7.2.6, "Tickets"
- Section 3.7.2.7, "ManyBalls"
- Section 3.7.2.8, "MiniColor"
- Section 3.7.2.9, "Chooser"
- Section 3.7.2.10, "HttpExample"
- Section 3.7.2.11, "HttpView"
- Section 3.7.2.12, "PushExample"

3.7.2.1 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.

3.7.2.2 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 = "j2me"
microedition.platform = "en-US"
microedition.platform = "IS08859_1"
```

3.7.2.3 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.

3.7.2.4 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.

3.7.2.5 Stock

Like the Http demonstration, this sample uses an HTTP connection to obtain information. Use the same preparation steps as Section 3.7.2.3, "Http".

Run the Demos project and launch the Stock MIDlet.

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

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 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. Stock tracker displays additional information about the selected stock, for example, the last trade and the high and low values.

Choose a stock and press Select.

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 main menu, select Alerts.

Select Add.

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

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

When the value is reached you 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.

3.7.2.6 Tickets

This demonstrates how an online ticket auction application might behave. The home screen displays a ticket ticker across the top. Click Done to continue to the Welcome To Tickets page. The Choose a Band field displays BootWare & Friends by default.

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 soft key or use the arrow keys to highlight Submit then press Select. 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 click the Save soft key. You are returned to the Choose a Band page. You can trigger the alert by making a bid that exceeds your alert value. Your settings determine how often the application checks for changes, so the alert may not sound for a few minutes.

To add a band 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. Choose the Save soft key. After confirmation 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.

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

userhome`\My Documents\NetBeansProjects\Demos\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 6. If desired, use the arrow keys and the select key to change these values. Choose the Save soft key.

3.7.2.7 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.

3.7.2.8 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.

3.7.2.9 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, then select. The left and right keys jump between lists. Up and down keys move item by item.

  Click OK to continue.

3.7.2.10 HttpExample

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

3.7.2.11 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.

3.7.2.12 PushExample

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

3.7.3 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 or choose 2 to calculate the result.

3.7.4 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 1, Start. The scrambled puzzle is displayed. The arrow keys move the empty space, displacing tiles accordingly (the arrow key indicates which tile to swap with the space). 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.

3.7.5 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.

3.7.5.1 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.

3.7.5.2 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.

3.7.6 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.

3.7.7 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.

3.8 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.

3.8.1 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.

3.8.2 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.

3.8.3 NetworkDemoIMPNG

This demo is a headless version of Section 3.7.5.1, "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
  ```

3.8.4 PDAPDemoIMPNG

This demo is a headless version of the PDAPDemo file browser in Section 17.3.1, "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

4 Creating and Editing Projects

A project is a group of files comprising a single application. Files include source files, resource files, XML configuration files, automatically generated Apache Ant build files, and a properties file.

When a project is created, the SDK performs these tasks:

- Creates a source tree you can examine in the Section 4.3, "Working With Projects" or Section 4.4, "View Project Files".
- Sets the emulator platform for the project.
- Sets the project run and compile-time classpaths.
- Creates a build script that contains actions for running, compiling, debugging, and building Javadoc. Project properties influence the build process, as described in Section 5.4, "Building a Project". See also Section 14.4, "Build a Project from the Command Line".

Java ME SDK and NetBeans create their project infrastructure directly on top of Apache Ant. Java ME SDK projects can be opened and edited in NetBeans, and vice-versa. With the Ant infrastructure in place, you can build and run your projects within the SDK or from the command line.

The SDK provides two views of the project:

- The Projects window provides a logical view of the project.
- The Files window displays a physical view of the project.

Project settings are controlled in the project Properties window. Typically, you right-click on an item or subitem in a tree (a project, a file, or a device) and select Properties.

4.1 Project Types

The CLDC/MIDP platform implements the Mobile Information Device Profile and Connected Limited Device Configuration (JSRs 118 and 139). As described in Section 2.1.2, "IMP-NG", the IMP-NG platform is a type of CLDC platform.

The CDC platform is implemented to support Advanced Graphics and User Interface Optional Package for the J2ME Platform, Personal Basis Profile 1.1, and the Connected Device Configuration (JSRs 209, 217 and 218). The AGUI API combines the PBP API and a subset of Java Platform, Standard Edition (Java SE) Swing capabilities.

4.1.1 CLDC Projects

A MIDP application (a MIDlet), is deployed as a MIDlet suite. A MIDlet suite is distributed as a Java archive (JAR) file and a Java Application Descriptor (JAD) file.

The IMP-NG version of a MIDlet is an IMlet. However, because IMP-NG is a subset of CLDC you can write IMP-NG applications as MIDlets.

The JAR file includes the Java classes for each MIDlet in the suite, Java classes shared between MIDlets, resource files, and other supporting files. The JAR file also includes a manifest describing the JAR contents and specifying attributes the Application Management Software (AMS) uses to identify and install the MIDlet suite.

The JAD file contains attributes that allow the AMS to identify, retrieve, and install the MIDlets in a project. The SDK automatically creates JAD and JAR files when you build the project.

4.1.2 CDC Projects

The CDC platform is implemented to support Advanced Graphics and User Interface Optional Package for the J2ME Platform, Personal Basis Profile 1.1, and the Connected Device Configuration (JSRs 209, 217 and 218). The AGUI API combines the PBP API and a subset of Java Platform, Standard Edition (Java SE) Swing capabilities.

Oracle Java ME SDK version 3.2 supports CDC projects running as standalone applications. The CDC project structure and behavior are much the same as that of CLDC projects.

Note: An Xlet cannot be run standalone. It depends upon an application manager to manage its life cycle (its state) and system services. Xlets are not supported in this release.

A standalone CDC project requires a main application class that includes a method named `main()` that handles class loading, object creation, and method execution. The application interacts directly with the Java runtime environment to manage its own life cycle and system resource needs. When the `main()` method exits, the standalone application terminates.

4.2 The Project Wizard

This section describes how to use the Project Wizard to create Java ME projects. The project provides a basic infrastructure for development. You provide source files, resource files, and project settings as needed. Most project properties can be edited later. For more on project properties, see Chapter 5, "Viewing and Editing Project Properties".

4.2.1 Create a CLDC Project

Follow these steps to create a MIDP project.

1. Select File > New Project.

   The New Project wizard opens.

2. In the Choose Project window, select the Java ME category, and the Mobile Application project type. Click Next.

3. On the Name and Location page, specify a project name. Most of the form is auto-filled, but you can alter any of the editable fields.

   Checking Create Hello MIDlet inserts sample MIDlet code as a template for your application. Click Next.

4. On the Select Platform page, be sure to **change the platform selection** to CLDC Oracle Java(TM) Platform Micro Edition SDK 3.2. The platform determines which devices you see in the Device dropdown menu.

   Make a device selection. The selected device typically determines the Device Configuration. **If more than one Device Profile is available, be sure to make a selection**.

   Click Finish.

4.2.2 Create an IMP-NG Project

The process for creating an IMP-NG project is almost the same as a CLDC project. There are just a few things to watch for.

1. Select File > New Project.

   The New Project wizard opens.

2. In the Choose Project window, select the Java ME category, and the Mobile Application project type. Click Next.

3. On the Name and Location page, specify a project name. Most of the form is auto-filled, but you can alter any of the editable fields.

   Do not check Create Hello MIDlet. The Hello MIDlet sample code does not work for headless devices because it assumes a display is available.

   Click Next.

4. On the Select Platform page, be sure to **change the platform selection** to CLDC Oracle Java(TM) Platform Micro Edition SDK 3.2. The platform determines which devices you see in the Device dropdown menu.

   Choose an IMPNG device, and choose a Device Profile.

   Click Finish.

5. To run the new project follow the steps in Section 3.2, "Running a Project".

4.2.3 Create a CDC Project

NetBeans provides a wizard for creating new projects quickly and easily. Most project properties can be edited later on.

1. Select File > New Project.

   The New Project wizard opens.

2. In the Categories window choose Java ME and in the Projects window choose CDC Application.

3. On the Name and Location page, specify a project name. Most of the form is auto-filled, but you can alter any of the editable fields. Click Next.

4. On the Select Platform page, be sure to **change the platform selection** to CDC Oracle Java(TM) Platform Micro Edition SDK 3.2. The platform determines which devices you see in the Device dropdown menu.

Choose a CDC device, and choose a Device Profile.

Click Finish.

5. To run the new project follow the steps in Section 3.2, "Running a Project", except you can select your new project instead of a sample project.

4.2.4 Import a Legacy MIDP Project

If you created a project using the Sun Java Wireless Toolkit for CLDC on Windows or Linux you can import your MIDlets into Java ME SDK projects. You can also use this procedure to create a project based upon a legacy sample project.

1. Select File > New Project.
2. In the Projects area, select Import Wireless Toolkit project. Click Next.
3. Specify the WTK project location.

   Use browse to open the directory containing the legacy project.
4. Select a project and click Next.
5. Supply the Project Name, Location, and Folder for the new project.

   The default name, project name and folder name are based on the name of the project you are importing. Click Next.
6. Select the Platform type, the default device, and the configuration and profile, if applicable. Click Finish.

   Your new project opens in the Projects window.
7. If the legacy project used signing, you must configure the signing properties as described in Section 13.4, "Managing Keystores and Key Pairs".

4.2.5 Import a Legacy CDC Project

If you created a project using the CDC Toolkit, you can import your applications into Java ME SDK projects. You can also use import to create a project based upon a sample project.

Note: Standalone projects created in the CDC Toolkit can be imported. Xlets cannot be imported.

The CDC platform name for the Oracle Java ME SDK version 3.2 does not match the legacy platform name in the CDC Toolkit 1.0 and the CDC Mobility Pack. Consequently, you get a reference error when you import a legacy CDC project.

Note: To avoid the reference error, create a platform with the legacy name, as described in Section 2.2.2, "Create a Platform for Legacy CDC Projects". You only need to create the platform once.

1. Select File > New Project.
2. In the Projects area select Import CDC Toolkit Project. Click Next.
3. Browse to select the project location.

The wizard detects any applications in the legacy installation and displays their locations on disk. Select a project and click Next.

4. Supply the Project Name, Location, and Folder for the new project. Note, the default name project name and folder name are based on the name of the project you are importing. Click Finish.

   The imported project opens in the Projects window.

   See also: Section 2.2.2, "Create a Platform for Legacy CDC Projects", Section 4.4, "View Project Files".

4.3 Working With Projects

The logical view of the project, shown in the Projects window, provides a hierarchy of sources and resources. Right-click on the project node to see actions related to the project.

New. Opens a form to build a new object for the current project. The new object is placed in the project's file structure by default, but you can control the file name and location. The possible objects are dependent on the currently selected project. For example, if the project is CLDC, the options are MIDlet, Java class, Java package, or Java interface. The New > Other option permits adding different types of files to the project. For a sample procedure, see Section 20.1, "Generating Stub Files from WSDL Descriptors".

Build. Builds a distribution Java archive (JAR) file. Project properties control the build process as described in Section 5.4, "Building a Project".

Clean & Build. Cleans, then builds a distribution JAR file.

Clean. Cleans the build files.

Generate Javadoc. See the online help topic Generating Javadoc Documentation.

Deploy. See the online help topic "*Deploying Java ME Applications*".

Batch Build..., **Batch Clean & Build...**, **Batch Clean...**, **Batch Deploy...** See the online help topic "*About Java ME MIDP Projects*".

Run. Runs the project with the default device, as specified on the Platform property page. See Section 5.2, "Platform Selection".

Run With... Run the selected project with a device you choose. This option can override the default device specified in the project properties.

Debug. See the online help topic "*Debugging Tasks: Quick Reference*".

Profile. Attach the profiler to the selected project. See Chapter 9, "Profiling Applications".

Set as Main Project. Make the current project the new main project. Toolbar actions, such as clicking the green Run button, act upon the main project by default.

Unset as Main Project. This option is visible if the selected project is already the main project.

Open Required Projects. Open any projects that the current project requires.

Close. Close the current project. Be sure that any processes are stopped, as closing a project might not stop the emulator.

4.4 View Project Files

The Files window displays a physical view of all project files. Right-click to view project properties or choose an action related to the project.

build. The output directory for the compiled classes listed below. This directory also contains `manifest.mf`, the manifest file that is added to the JAR file.

- `compiled`. Contains all compiled classes.
- `obfuscated`. Holds the obfuscated versions of the class files.
- `preprocessed`. Holds the source files after they are preprocessed. The files differ from the original source files if you are using project configurations.
- `preverified`. Holds the preverified versions of the class files. These files are packaged into your project's distribution JAR.
- `preverifysrc`. Versions of the source files before they are preverified.

dist. The output directory of packaged build outputs (JAR files and JAD files). The `dist` directory also contains generated Javadoc documentation.

lib. Contains libraries you have added to the project. See Section 5.4.3, "Adding Libraries and Resources".

nbproject. The directory that contains the project Ant script and other metadata. This directory contains the following files:

- `build-impl.xml`. The SDK-generated Ant script. Do not edit `build-impl.xml` directly. Always override its targets in `build.xml`.
- `private/private.properties`. Properties that are defined for you alone. If you are sharing the project, any properties you define in this file are not checked in with other project metadata and are only applied to your SDK installation.
- `project.properties`. Ant properties used to configure the Ant script. This file is automatically updated when you configure the project's properties. Manual editing is possible, but it is not recommended.
- `project.xml` and `genfiles.properties`. Generated metadata files. It is possible to edit `project.xml` manually, but it is not recommended. Do not edit `genfiles.properties`.

res. Resource files you have added to the project. See Section 5.4.3, "Adding Libraries and Resources".

src. The project source files.

build.xml. The build script. This build script only contains an import statement that imports targets from `nbproject/build-impl.xml`. Use the `build.xml` to override targets from `build-impl.xml` or to create new targets.

See also: Section 4.2.1, "Create a CLDC Project", and Section 4.2.3, "Create a CDC Project".

4.5 Create a New MIDlet

To create a new MIDlet from the Files view, right-click a project and select New > MIDlet. With this form you can specify the name of the MIDlet and its location within the selected project.

MIDlet Name. The name that users see when the application runs on a device.

MIDlet Class Name. The name of the new MIDP class.

MIDlet Icon. The path to an icon associated with the MIDlet. Users see the icon when the application runs on a device.

Project. Displays the name of the project.

Package. Specifies the location of the MIDlet class in the package hierarchy. You can select an existing package from the drop down menu, or type in the name of a new package. The new package is created along with the class.

Created File. Displays the name and the location of the MIDlet in the system's hierarchy.

When the new MIDlet is created the SDK automatically adds it to the project's Application Descriptor File.

4.6 Add Files to a Project

For all projects, right-click to use the context menu to add files to a project. Using this method places files in the proper location in project source or resources.

To add a MIDlet, Java class, Java package, Java interface or Other files, right-click the project name or the Source Packages node, choose New, and select the file type.

To add files by format (Project, JAR, Folder, Library) right-click the Resources node and select a format. See Section 5.4.3, "Adding Libraries and Resources".

It is possible to add files by copying them directly to the project directory but it is not recommended.

4.7 Search Project Files

To search a project's files, right-click on the project and select Find...

The Find in Files utility supports searching a project's file contents or file names. The search input fields supports simple text matching and regular expressions.

Containing Text. The string you are looking for. If File Name Patterns is empty, all files are searched.

File Name Patterns. The files you are searching in. If the Containing Text field is empty you get a listing of files that match the pattern.

The options Whole Words, Match Case, and Regular Expression further restrict the search. Regular Expression Constructs are fully explained in:

`http://download.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html#sum`

4.8 Debugging CLDC and IMP-NG Projects

Java ME Projects use standard NetBeans debugging utilities. Please refer to the NetBeans help topic *Debugging Tasks: Quick Reference*.This topic includes links to a variety of debugging procedures.

If you have an external device that runs a supported runtime you can perform on-device debugging. The device must be detected by the Device Selector, as described in Section 6.2, "Adding a Real Device".

5 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 5.1, "General Project Properties"
- Section 5.2, "Platform Selection"
- Section 5.3, "Editing Application Descriptor Properties"
- Section 5.4, "Building a Project"
- Section 5.5, "Running Settings"

5.1 General Project Properties

To view the General property page, right-click on a project, choose Properties, and select the General category. The general properties page displays basic project properties. You can set application versioning here, but all other values cannot be edited.

The project name, folder, and source location are set when the project is created. The Application Version Number field displays the version number of the current build.

Application Versioning

The Application Version Counter field displays the next version number to be used. The default advance is 0.0.1. To advance the number beyond this, use the dropdown menu to select a new digit, or enter the value into the field. For example, changing the value to 3 results in a build number of 0.0.3. Changing the value to 100 results in the version number 0.1.0.

Required Projects

This area displays projects you have added to this project. It might be a dependent project or an external library. See Section 5.4.3, "Adding Libraries and Resources".

5.2 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 Platform category.

Select a platform type from the dropdown menu.

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.

Be sure that a Device Profile is selected, and select the optional packages you want to include in this project. The selected APIs are automatically added to the project's classpath. See Section 4.2.1, "Create a CLDC Project".

5.3 Editing Application Descriptor Properties

To view this property page, right-click on a project, choose Properties, and select the Application Descriptor category. The Application Descriptor properties page enables adding, editing, or deleting project attributes.

5.3.1 CDC Attributes

To view this property page, right-click on a CDC project and choose Properties. Select the Application Descriptor category.

Application Name. The display name of the application on the target device.

ApplicationVendor. The company name or author name for the application.

Description. A concise description of the application.

Detail Description. A detailed description of the application.

5.3.2 MIDP Attributes

To view this property page, right-click on a MIDP project and choose Properties. Select the Application Descriptor category, and select the Attributes tab.

The General Attributes table lists the attributes currently contained in the JAD and JAR manifest files:

Type. Lists whether the attribute is required or optional. Custom attributes for passing parameters to the MIDlet using the JAD are also available.

Name. The name of the attribute.

Value. The values for each attribute.

To avoid errors in verification:

- Define all required attributes.
- Do not begin user-defined attribute keys with `MIDlet-` or `MicroEdition-`.

5.3.2.1 Add an Attribute

Follow these steps to add an attribute.

1. Click Add... to open the Add Attribute window.
2. Choose an attribute from the Name combo box, or delete the current entry and add your own custom entry.

Note: Do not begin a user-defined attribute name with `MIDlet-` or `MicroEdition-`.

3. Enter a value for the attribute.
4. Click OK.

5.3.2.2 Edit an Attribute

1. Select an attribute.
2. Click Edit... to open the Edit Attribute window.
3. Enter a value for the attribute.
4. Click OK.

API permissions, Push Registry Entries, and API Permissions have their own property pages.

5.3.2.3 Remove an Attribute

Select an Attribute and click Remove to delete it from the list.

5.3.3 MIDlets

To view this page, right-click on a project and choose Properties. Select the Application Descriptor category, and select the MIDlets tab.

The MIDlets table lists the MIDlets contained in the suite and the following properties:

Name. The displayable name of the MIDlet that the user sees when the MIDlet is run on a mobile device.

Class. The Java class for the MIDlet.

Icon. An icon (a `.png` file), representing the MIDlet that the user sees when the MIDlet is run on a mobile device.

5.3.3.1 Add a MIDlet

1. Click Add... to open the Add MIDlet window.

   The window lists the MIDLets available in the project.
2. Enter a name, then select a MIDlet class from the dropdown menu.

   You can also choose an icon for the MIDlet from the MIDlet icon dropdown menu.
3. Click OK.

5.3.3.2 Edit a MIDlet

1. Select a MIDlet.
2. Click Edit... to open the Edit MIDlet window.
3. Enter a value for the attribute.
4. Click OK. The revised values are listed in the table.

5.3.3.3 Remove a MIDlet

Select a MIDlet and click Remove to delete it from the list.

5.3.3.4 Change MIDlet Display Order

The list order determines the order in which the MIDlets are displayed.

Select a MIDLet and select Move Up or Move Down to change its position.

5.3.4 Push Registry

To view this page, right-click on a project and choose Properties. Select the Application Descriptor category, and select the Push Registry tab.

See also Section 5.3.4.1, "Add a Push Registry Entry", Section 5.3.4.2, "Enabling a Push Registry Entry", Section 5.3.4.3, "Remove a Push Registry Entry", and Section 5.3.4.4, "Change Push Registry Display Order".

5.3.4.1 Add a Push Registry Entry

1. Click Add... to open the Add Push Registry window.
2. Enter Class Name, Sender IP, and Connection String values.
   - **Class Name**. The MIDlet's class name.
   - **Sender IP**. A valid sender that can launch the associated MIDlet. If the value is the wildcard (*), connections from any source are accepted. If datagram or socket connections are used, the value of Allowed Sender can be a numeric IP address.
   - **Connection String**. A connection string that identifies the connection protocol and port number.
3. Click OK.

   The new values are listed in the table. A push registration key is automatically generated and shown as an attribute in the MIDlet suite's Java Application Descriptor (JAD) file.

5.3.4.2 Enabling a Push Registry Entry

To make use of the Push Registry, you must also have permission to access the Push Registry API, `javax.microedition.io.PushRegistry`. API permission, are handled in the API Permissions property page (Section 5.3.5, "API Permissions").

5.3.4.3 Remove a Push Registry Entry

Select an entry and click Remove to delete it from the list.

5.3.4.4 Change Push Registry Display Order

The list order determines the order in which the MIDlets are displayed. Select an entry and select Move Up or Move Down to change its position.

5.3.5 API Permissions

These properties set permission attributes for protected APIs called by the MIDlet suite. To view this property page, right-click on a project and choose API Permissions. Select the Application Descriptor category, and select the Attributes tab.

See Section 5.3.5.1, "Adding Permission Requests".

5.3.5.1 Adding Permission Requests

1. Click the Add Button.

The API Permission for API dialog opens.

2. Choose an API from the dropdown list or enter an API into the combo box and click OK.
3. (Optional) In the Requested Permissions table, check the Required box if you want installation to fail when that permission cannot be granted.

For more information, see *Security for MIDP Applications* in the MIDP 2.0 (JSR 118) specification, available at:
`http://developers.sun.com/techtopics/mobility/midp/articles/pushreg/`.

5.4 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 5.4.2, "Compiling"
- Section 5.4.3, "Adding Libraries and Resources"
- Section 5.4.5, "Obfuscating"
- Section 5.4.4, "Creating JAR and JAD Files (Packaging)"
- Section 5.4.6, "Signing"

5.4.1 Configuring Ant

To view this form, select Tools > Options, select Miscellaneous, and click the Ant tab.

Ant Home. The installation directory of the Ant executable the SDK uses. To change Ant versions, type the full path to a new Ant installation directory in this field or click Browse to find the location. You can only switch between versions 1.5.3 and higher of Ant. The Ant installation directory must contain a `lib` subdirectory which contains the `ant.jar` binary. If you enter a directory that does not match the expected structure, the SDK gives you an error. See the NetBeans help topic for more options.

Save All Modified Files Before Running Ant. If selected, saves all unsaved files in the SDK before running Ant. It is recommended to leave this property selected because modifications to files in the SDK are not recognized by Ant unless they are first saved to disk.

Reuse Output Tabs from Finished Processes. If selected, writes Ant output to a single Output window tab, deleting the output from the previous process. If not selected, opens a new tab for each Ant process.

Always Show Output. If selected, the SDK displays the Output window for all Ant processes. If not selected, raises the Output window tab only if the Ant output requires user input or contains a hyperlink. Output that contains hyperlinks usually denotes an error or warning.

Verbosity Level. Sets the amount of compilation output. Set the verbosity lower to suppress informational messages or higher to get more detailed information.

Classpath. Contains binaries and libraries that are added to Ant's classpath. Click Add Directory or Add JAR/ZIP to open the Classpath Editor.

Properties. Configures custom properties to pass to an Ant script each time you call Ant. Click Manage Properties to edit the properties in the property editor. This property is similar to the Ant command-line option, `-Dkey=value`. The following default properties are available:

${build.compiler.emacs}. Setting this property to true enables Emacs-compatible error messages.

5.4.2 Compiling

To view this property page, right-click on a project and choose Properties. In the Properties window Build category, choose Compiling.

This page enables you to set the following options:

Generate Debugging Info. If checked, the compiler generates line numbers and source files information. This is the `-g` option in `javac`. If unchecked, no debugging information is generated (the `-g:none` option in `javac`).

Debug Block Level. The block level can be set to: debug, info, warn, error, and fail.

Compile with Optimization. If checked, the compiled application is optimized for execution. This is the `-O` option in `javac`. Optimizing can slow down compilation, produce larger class files, and make the program difficult to debug.

Report Uses of Deprecated APIs. If checked, the compiler lists each use or override of a deprecated member or class. This is the `-deprecated` option in javac. If unchecked, the compiler shows only the names of source files that use or override deprecated members or classes.

Encoding. Overrides default encoding used by preprocessor and compiler. The default value is the default encoding used by your VM.

5.4.3 Adding Libraries and Resources

To view this property page, right-click on a project and choose Properties. In the Properties window Build category, choose Libraries and Resources to add a dependent project, libraries, and other supporting files to the current project.

Add Project. A JAR file produced by another project and the associated source files and Javadoc documentation. Adding this item to a classpath sets up a dependency between the current project and the selected JAR file.

Add Library. A Library is a collection of JAR files or folders with compiled classes, which can optionally have associated source files and Javadoc documentation. If the Package checkbox is checked the library is included in the application's JAR file. If it is not checked, the library is copied into the `lib` directory.

Add JAR file. A JAR file created by another project.

Add Folder. The root of a package or directory containing files.

When a library or resource is added, it is visible in the Libraries and Resources table, which reflects the order of the libraries and resources in the classpath. To change the order in the classpath, select the listing and click Move Up or Move Down. You can also remove libraries and resources from this page.

Each row in the table has a Package check box. If Package is checked, the library or resource is bundled and added to the project JAR file. If Package is not checked, the library or resource is copied to the `lib` subdirectory at build time.

5.4.4 Creating JAR and JAD Files (Packaging)

To view this property page, right-click on a project and choose Properties. In the Properties window Build category, choose Creating JAR.

You can set the following options:

JAD File Name. Name of the JAD file created by the project sources. The file name must have a `.jad` extension.

JAR File Name. Name of the JAR file created by the project sources. The file name must have a `.jar` extension.

Compress JAR. If checked, the JAR file is compressed.

5.4.5 Obfuscating

To view this property page, right-click on a project and choose Properties. In the Properties window Build category, choose Obfuscating.

Use the Obfuscation properties page to set the level of obfuscation for project files.

Move the Obfuscation slider to set the level. The Level Description window describes the impact each level has.

You can add more obfuscation parameters in the Additional Obfuscation Settings window.

5.4.6 Signing

To view this property page, right-click on a project and choose Properties. In the Properties window Build category, choose Signing. These properties allow you to enable signing and assign key pairs to a CLDC project. See Section 13.1, "Security Domains".

Sign Distribution. Check this box to enable signing for the MIDlet suite. If it is unchecked, this page is disabled.

Keystore. A file that stores one or more key pairs as a keystore (`.ks`) file. The dropdown menu lists all available keystores. Click the Unlock button to unlock a keystore with the keystore password.

Alias. A name assigned to a key pair within a keystore. The dropdown menu lists the aliases available for the selected keystore. Click the Unlock button to unlock a key pair for use.

The Certificate Details window provides information about the certificate assigned to the key pair.

Click Open Keystores Manager to manage keystores and key pairs. See Section 13.4, "Managing Keystores and Key Pairs" and Section 5.4.8, "Exporting a Key".

Note: CDC projects cannot be signed with the Signing tool. See Section 5.4.7, "Signing CDC Projects".

5.4.7 Signing CDC Projects

To sign a CDC project use the JDK `jarsigner` command from the command line. For example:`jarsigner.exe -keystore keystore.ks -storepass keystorepwd MyCdcApp.jar dummyCA`

5.4.8 Exporting a Key

Follow these steps to export a key into an emulator:

- Select Tools > Keystores. This opens the Keystores Manager.

  You can use the Keystores Manager to add a keystore to the Keystores list. Click the Add Keystore button. After you create the keystore, click New to create a key pair.
- In the Keys area, select a key, and click Export. This opens the dialog Export Key into Java ME SDK/Platform/Emulator.
- Select the target emulator from the Emulator list.
- Select the Security Domain.
- Click Export to export your key pair to the selected emulator.

  Your key is added to the bottom of the list in Keys Registered in the Emulator.

The Export window has the following components:

Keystore File. Displays the name of the keystore file to which the key pair belongs. This field cannot be edited.

Key Pair Alias. The name given to the key pair within the keystore. This field cannot be edited.

Certificate Details. Displays the details of the certificate of the key to be exported.

Emulator. The drop-down menu lists all the device emulators available. See Section 13.1, "Security Domains".

Security Domain. Enables you to select a security domain for the key pair. The menu lists all domains supported by the selected emulator platform.

Keys Registered in the Platform. Lists all keys that have been registered in the selected platform. Click to select the key you want to export.

Delete Key. Deletes a selected key from the Keys Registered in the Emulator window.

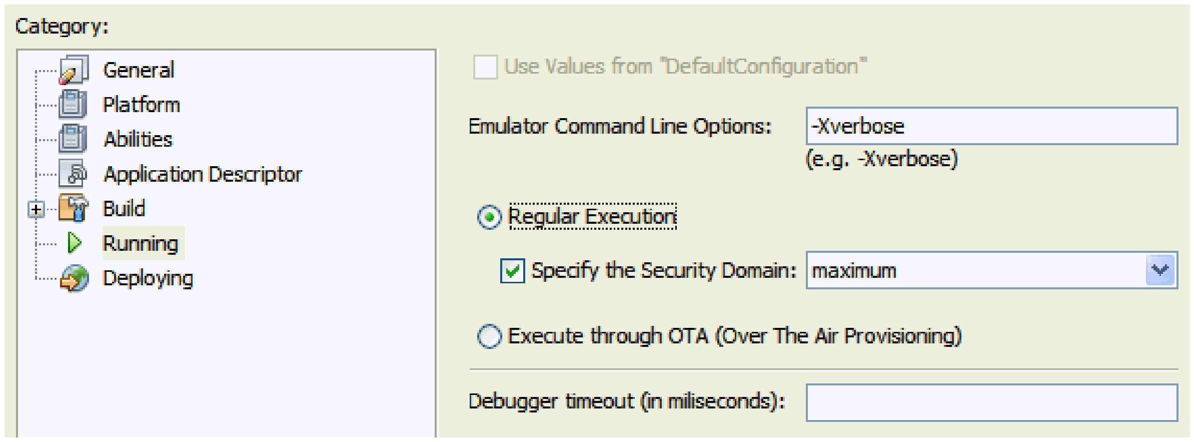
Export. Exports the selected key to the selected emulator. The export button is enabled if it is possible to export the key. If a specific key is installed it cannot be installed again.

5.5 Running Settings

To view this property page, right-click on a project and choose Properties. In the Properties window, choose Running. The options shown depend on the platform. See Section 5.5.1, "MIDP Project Run Options".

5.5.1 MIDP Project Run Options

To set emulator command line options for a MIDP project, type in the command line switches. See Section 14.3, "Emulator Command Line Options".



Use Values from "DefaultConfiguration"

Emulator Command Line Options: -Xverbose
(e.g. -Xverbose)

[✓ Regular Execution]

[✓] Specify the Security Domain: maximum

Execute through OTA (Over The Air Provisioning)

Debugger timeout (in miliseconds):

For CLDC projects, the Regular execution button is enabled by default. The setting for "Specify the Security Domain" applies when the project is run on an emulator. It does not apply for OTA provisioning or an external emulator platform.

If you do not check Specify the Security Domain the project runs with the default that was assigned when the project was created. If you check the box, you can select a domain from the dropdown list. See Section 13.1, "Security Domains" and Section 13.2.2, "Specify the Security Domain for a Project".

5.5.2 CDC Project Run Options

For CDC projects you must enter the name of the entry point Java file in the Main Class field. The Main Class Browse button only shows executable classes in the project's source folders. For a CDC project you see all classes with a static main method, or classes extending the Applet or Xlet classes.

Arguments are passed only to the main class, not to individual files. If an Xlet is executed, all arguments are passed to all the Xlets you specify.

For VM options, see Section 14.3.2, "CDC Options".

6 Working With Devices

The Oracle Java ME SDK emulator simulates a CLDC or CDC 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. To make your own skin, see Section 6.11, "Using the Custom Device Skin Creator".

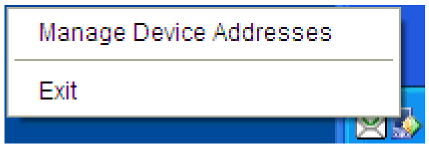
6.1 Emulating Devices

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 (Tools > Java ME > Device Selector).

6.1.1 The Device Manager on Windows

The SDK uses the device manager to detect devices and displays the available devices in the Device Selector window (Tools > Java ME > Device Selector). 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.



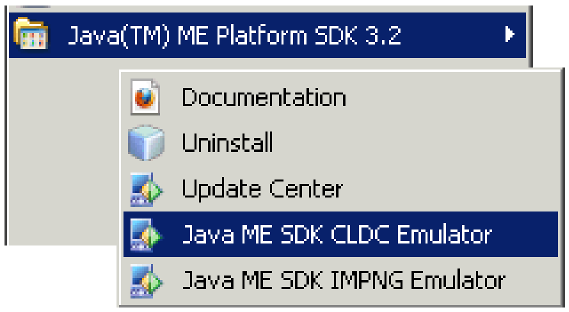
To restart the device manager, double-click *installdir*`\bin\device-manager.exe`. You can also start it from the command line as described in Section 14.1, "Run the Device Manager".

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.

6.1.2 Starting an Emulator

Typically an emulator is launched when a Java ME SDK project is run from the NetBeans IDE or the command line. The default emulator is determined by the Java ME platform selected for the project, as described in Section 2.2.1, "Java ME Platforms".

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.

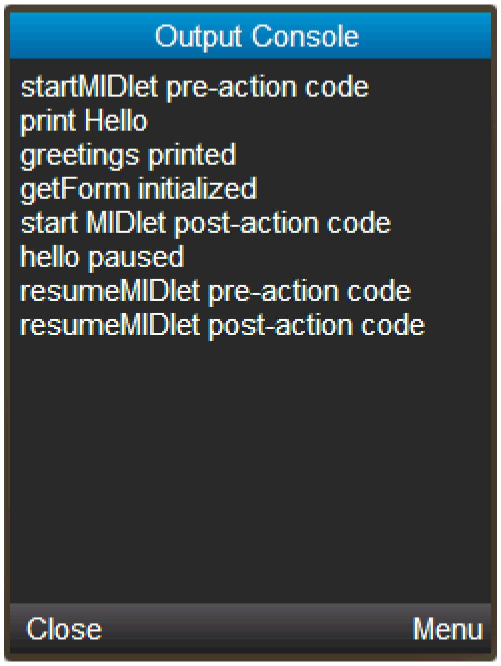
6.1.3 CLDC Application Management Software Home

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 6.5, "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 6.9, "Emulator Features" and Section 6.10, "Emulator Menus".

6.2 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-D, 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 6.1.1, "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.

6.3 Viewing Device Properties

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

If no Java ME platform is registered in NetBeans, the Device Selector displays a node labeled No Device Found. 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. To make your own device, see Section 6.11, "Using the Custom Device Skin Creator".

For Device names, see Section 8.4, "Oracle Java ME SDK Directories". The properties for each device skin are stored in XML files in your user work directory. See Table 8–1.

See also: Section 6.3.1, "Platform Properties", Section 6.3.2, "Device Information", and Section 6.3.3, "Device Properties"

6.3.1 Platform Properties

To view platform properties from the device selector, click on the platform node (for example, CLDC or CDC). The Properties window is, by default, docked in the upper right portion of the user interface. If the Properties window is not visible, select Windows > Properties.

To view the platform properties in a separate window, right-click on the platform node and select Properties. The information in the docked properties window and the separate window is the same.

6.3.2 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.

6.3.3 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. If the Properties window is not visible, select Windows > Properties.

6.4 Setting Device Properties

In the Device Selector Window, right-click on a device and select Properties. Any properties shown in gray font cannot be changed. You can adjust the device properties shown in black. Only CLDC options can be adjusted. The CDC options cannot be changed.

6.4.1 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 applications's objects. The Heapsize property is the maximum heap size for the emulator. You can choose a new maximum size from the dropdown menu.

Security Domain. Select a security setting from the dropdown menu. See Section 13.2.1, "Specify the Security Domain for an Emulator". Applies to CLDC platforms.

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).

6.4.2 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. See Chapter 11, "Monitoring Memory".

6.4.3 SATSA

See Section 21.1, "Card Slots in the Emulator".

6.4.4 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`.

6.4.5 Bluetooth and OBEX

Bluetooth. See Section 18.1, "Setting OBEX and Bluetooth Properties".

6.5 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.

6.6 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.

6.7 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 14.3, "Emulator Command Line Options".

6.8 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 is visible in the SDK (select Window > Output > Output). 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 2, 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 1, 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.

6.9 Emulator Features

Figure 6–1, "Emulator Features" shows common emulator features available on emulators for the CLDC platform.

Device Name. Shown in the upper window frame. See Table 8–1, " 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.

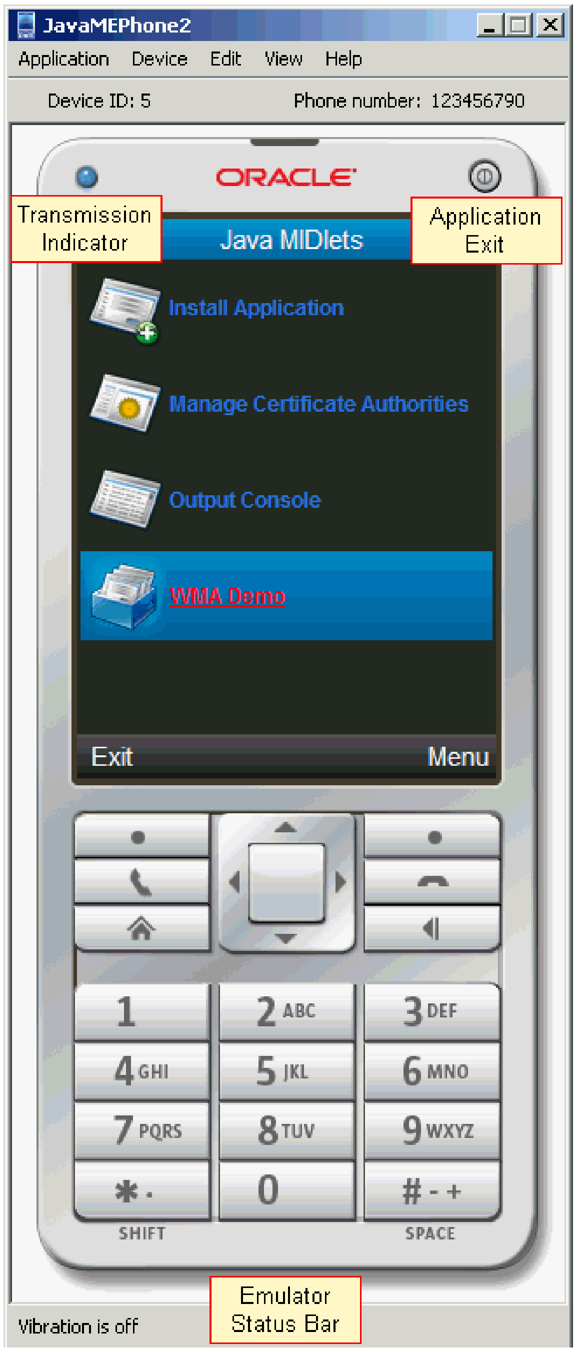
Menus. See Section 6.10, "Emulator Menus".

Device ID. See Table 8–1, " 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 6–1 Emulator Features



6.10 Emulator Menus

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

The emulator for the CDC platform has Application, View, and Help menus. The View and Help menus are the same on CDC and CLDC platforms. For CDC, the Device menu is not populated, and Application menu contains only the Exit option.

6.10.1 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 runing 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 6.8, "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).

6.10.2 Device

This menu is available on CLDC platforms only.

6.10.2.1 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 Netbeans, 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.

6.10.2.2 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.

6.10.2.3 Orientation

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.

6.10.2.4 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**. See Section 29.1, "Using ContactlessDemo".
- **File Connection**. Section 17.1, "FileConnection API".
- **Location**. Section 22.1, "Setting the Emulator's Location at Runtime".
- **Sensors**. Section 28.2, "Using a Mobile Sensor Project", Section 28.3, "Creating a Sensor Script File".

6.10.3 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.

6.10.4 View

The View menu is available for both CLDC and CDC platforms. 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.

6.10.5 Help

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

6.11 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.

6.11.1 Creating a New Custom Device Skin

Follow these steps to create a new custom device skin.

1. Select Tools > Java ME > Custom Device Skin Creator.

   The custom device tree displays Java ME platforms and custom devices, if any.

2. Select a platform and click the New... button.
3. Change the default configuration to match your specifications, and click OK.

Your skin is added to the custom device tree and eventually appears in the Device Selector. Your custom skin 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 custom device tree.

The skin definition is saved in *installdir*`\toolkit-lib\devices`.

6.11.2 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 completely deletes the skin.

7 Searching the WURFL Device Database

The Wireless Universal Resource File (WURFL) is an XML file that acts as a global database of mobile device capabilities. WURFL is an open source project at `http://wurfl.sourceforge.net/`. The WURFL DB (`http://www.wurflpro.com/`) is a web-based interface that allows WURFL contributors to add or change device information in the WURFL.

The SDK uses a WURFL module to discover devices based on API support or on physical characteristics such as physical memory size or display size.

7.1 WURFL Search for Devices

Follow these instructions to search for devices.

1. Select Tools > Java ME > Device Database Search.

   The WURFL Device Search tab opens in the main window.

2. Check Use Filter to see search options.

   If you do not check Use Filter, all devices in the database are listed. See Section 7.2, "WURFL Search Filtering".

3. Make a selection from the dropdown menu on the left.

   If applicable, the center dropdown displays a list of conditions. The menu on the right displays a value.

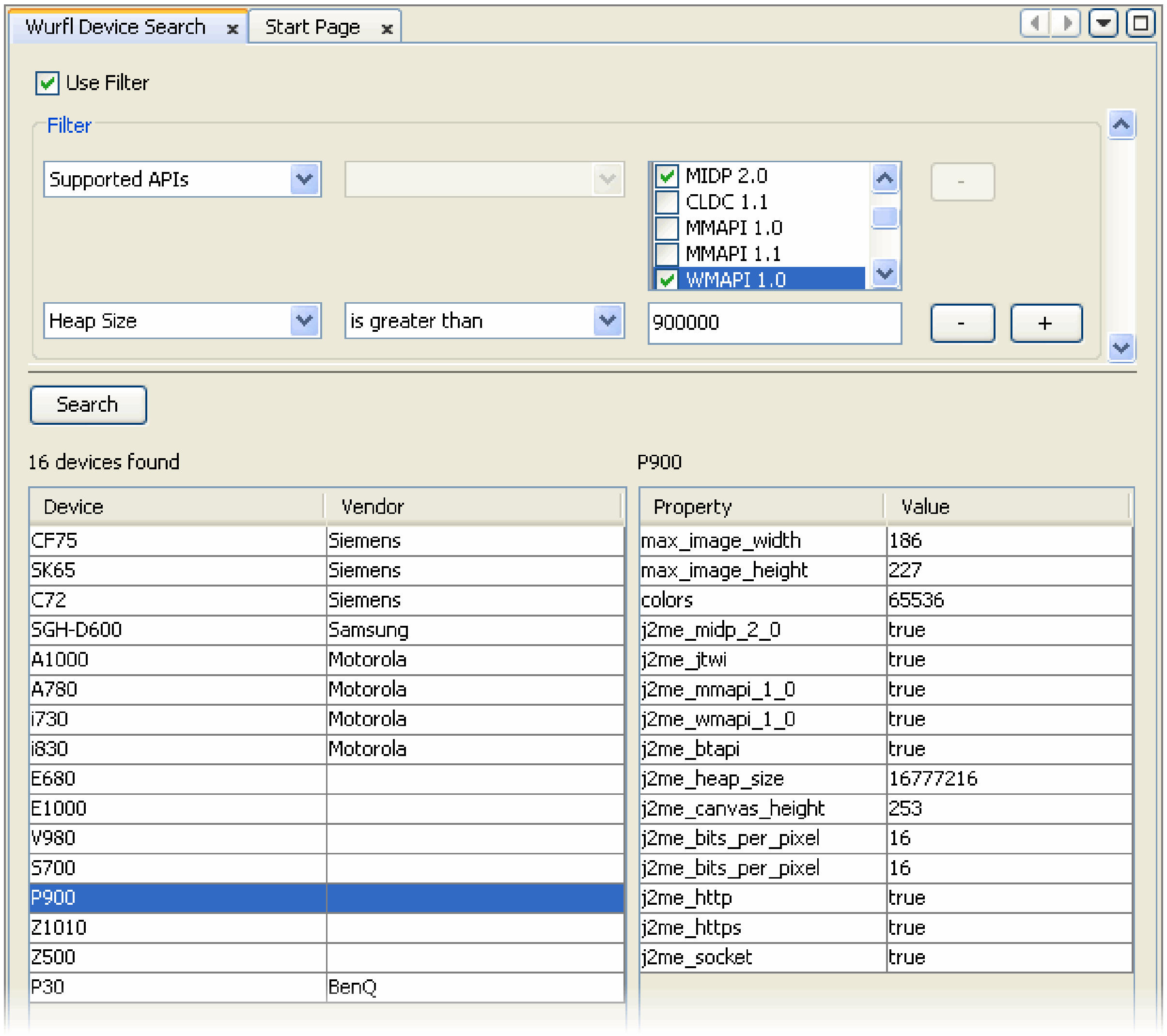
4. To add another search criteria, click the + button.

   Click the - button to remove a search setting.

5. Click the Search button.

   The search returns devices that match all the chosen criteria. The results are not case sensitive.

6. Click on a device to view its properties on the right, as shown below.



Use Filter [✓]

Supported APIs		MIDP 2.0 [✓], CLDC 1.1, MMAPI 1.0, MMAPI 1.1, WMAPI 1.0 [✓]	
Heap Size	is greater than	900000	

16 devices found

Device	Vendor
CF75	Siemens
SK65	Siemens
C72	Siemens
SGH-D600	Samsung
A1000	Motorola
A780	Motorola
i730	Motorola
i830	Motorola
E680	
E1000	
V980	
S700	
P900	
Z1010	
Z500	
P30	BenQ

P900

Property	Value
max_image_width	186
max_image_height	227
colors	65536
j2me_midp_2_0	true
j2me_jtwi	true
j2me_mmapi_1_0	true
j2me_wmapi_1_0	true
j2me_btapi	true
j2me_heap_size	16777216
j2me_canvas_height	253
j2me_bits_per_pixel	16
j2me_bits_per_pixel	16
j2me_http	true
j2me_https	true
j2me_socket	true

See Section 7.2, "WURFL Search Filtering".

7.2 WURFL Search Filtering

As discussed in Section 7.1, "WURFL Search for Devices", you can use the filter to set search constraints. If Use Filter is not checked all devices are listed. If Use Filter is checked, you must set at least one constraint.

Supported Properties

This utility searches on a predefined list of constraints that have corresponding properties in the Oracle Java ME SDK.

- Supported APIs

  You can check the APIs you want. Note, checking an API does not exclude APIs that are not checked.

  - MIDP 1.0, MIDP 2.0
  - CLDC 1.0, CLDC 1.1, MMAPI 1.0, MMAPI 1.1
  - WMAPI 1.0, WMAPI 1.1, WMAPI 2.0

  - Bluetooth API
  - 3D API
  - Location API
- Vendor
- Device
- Resolution Width/Height

  The device resolution.
- Maximum Image Width/Height

  The maximum image size that the device can display.
- Physical Memory Size

  The built-in memory size.
- Heap Size

  Memory limit in bytes at runtime.
- Number of Colors

  The number or colors the device's display supports.
- Supports Wi-Fi
- Supported Image Formats

  Check the image type. Unchecked types might still be supported.
  - bmp
  - jpeg
  - gif

To see the full list of WURFL constraints, go to:
`http://wurfl.sourceforge.net/help_doc.php`.

See also Section 7.1, "WURFL Search for Devices".

8 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 8.1, "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 8.2, "Installation Directories"
- Section 8.3, "NetBeans User Directories"

To locate logs, see Section 15.1, "Device Manager Logs", and Section 15.2, "Device Instance Logs".

8.1 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 6.1.1, "The Device Manager on Windows". A different user can then launch the SDK and own all processes.

8.2 Installation Directories

The SDK directory structure conforms to the Unified Emulator Interface Specification (`http://java.sun.com/j2me/docs/uei_specs.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 21.2, "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 14.2, "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 6.1, "Emulating Devices".
- `emulator`. UEI compliant emulator. See Section 14.3, "Emulator Command Line Options".
- `jadtool`. Tool for signing MIDlets. See Section 14.6.2, "Sign MIDlet Suites (jadtool)".
- `mekeytool`. Management of ME keystores. See Section 14.6.3, "Manage Certificates (MEKeyTool)".
- `preverify`. The Java ME preverifier.
- skin-creator.exe. Tool for creating new skins. See Section 6.11.1, "Creating a New Custom Device Skin".
- `wma-tool`. A command line tool for sending and receiving SMS, CBS, and MMS messages. See Section 23.3, "Running WMA Tool".
- `wscompile`. Compiles of stubs and skeletons for JSR 172. See Section 14.7, "Generate Stubs (wscompile)".

* `docs`. Release documentation.
* `legal`. License and copyright files.
* `lib`. JSR JAR files for compilation.
* `runtimes`. CDC, 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.

8.3 NetBeans User Directories

These are the default NetBeans user directories.

* NetBeans default project location:

  userhome`\My Documents\NetBeansProjects`

* To see the NetBeans user directory, select Help > About in the main window. The default location is:

  userhome`\.netbeans\`*version*

8.4 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\`*version*

- Device working directories

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

  The named subdirectories each correspond to an emulation device, as described in Table 8–1. Any detected real devices are also added to this directory space. Device detection is described in Section 6.2, "Adding a Real Device."

Table 8–1 Device Names

Device	Platform	Emulator #
ClamshellJavaMEPhone1	CLDC	0
DefaultCdcPbpPhone1	CDC	1
IMPNGPhone1	CLDC	2
IMPNGPhone2	CLDC	3
JavaMEPhone1	CLDC	4
JavaMEPhone2	CLDC	5
VgaAGUIPhone1	CDC	6
VgaCdcPhone1	CDC	7

- Device instances (device definitions).

  installdir`\toolkit-lib\process\device-manager\device-adapter`

  This directory contains the bean files for the adapter categories. The beans in this directory and subdirectories determine whether a skin is visible in the Device Selector, among other things. You should not manipulate these files directly.

  See Section 6.11, "Using the Custom Device Skin Creator" for instructions on creating your own custom skin.

- Both default skins and custom skins created with the Custom Device Skin Creator are represented in the device-adapter directory.

Note: Do not manipulate custom skin files from the operating system. All custom skin activity should take place in the Custom Device Skin Creator.

9 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. After you close the emulator you can export the data to a `.nps` file you can load and view later. As you view the snapshot you can investigate particular methods or classes and save a customized snapshot (a `.png` file) for future reference.

You can start a profiling session from the NetBeans IDE, as described in Section 9.1, "Collecting and Saving Profiler Data in the IDE", or from the command line, as discussed in Section 14.3.4, "Command Line Profiling". It is important to understand that profiling data produced from the command line has a different format (`*.prof`) than data produced from the NetBeans profiler (a `.nps` file).

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

Profiling data from Oracle Java ME SDK projects is displayed in a tab in the IDE. The NetBeans IDE has a Profiling window (Window > Profiling > Profiler) but it is not discussed here. Because only performance profiling is supported, the Profiler window has limited usefulness for Java ME applications.

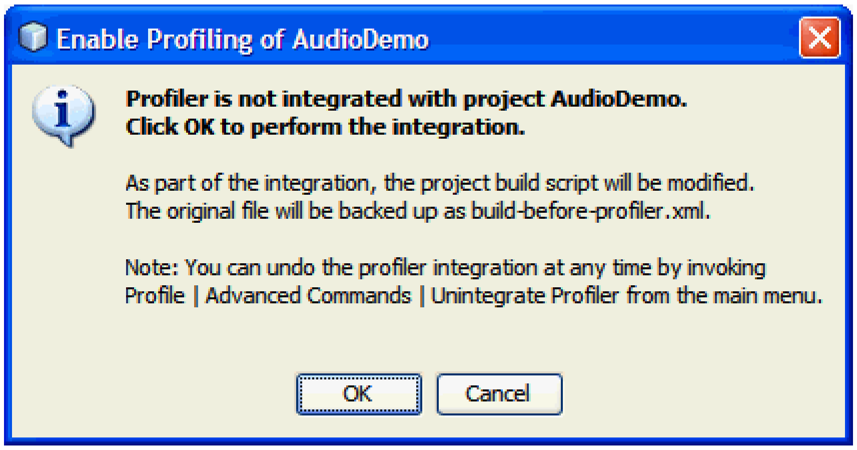
9.1 Collecting and Saving Profiler Data in the IDE

This procedure describes interactive profiling. (To run profile an application from the command line, see Section 14.3.4, "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 6.4, "Setting Device Properties".

1. In the Projects widow, right-click on the project you want to profile and select Profile.

   If this is the first time profiling this project you are prompted to integrate the profiler. Click Yes to perform the integration.

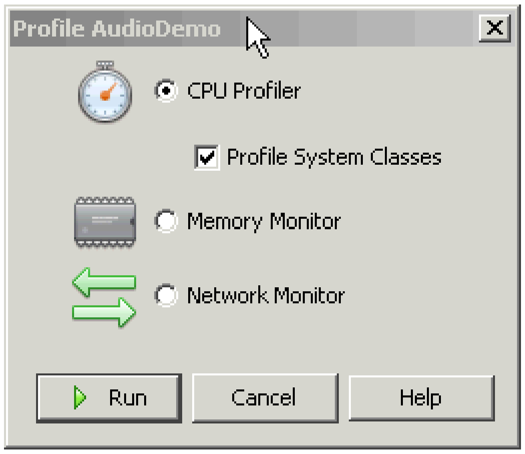

Enable Profiling of AudioDemo

Profiler is not integrated with project AudioDemo.
Click OK to perform the integration.

As part of the integration, the project build script will be modified.
The original file will be backed up as build-before-profiler.xml.

Note: You can undo the profiler integration at any time by invoking
Profile | Advanced Commands | Unintegrate Profiler from the main menu.

The profiler attaches. You are prompted for the running options.


Profile AudioDemo

[●] CPU Profiler

[✓] Profile System Classes

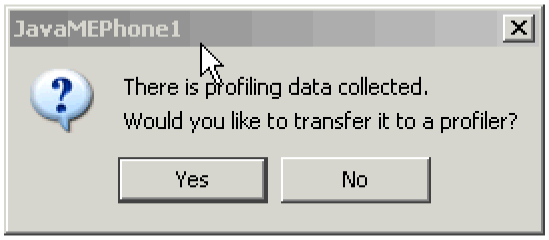
○ Memory Monitor

○ Network Monitor

Choose the CPU Profiler, and optionally check Profile System Classes. Press Run.

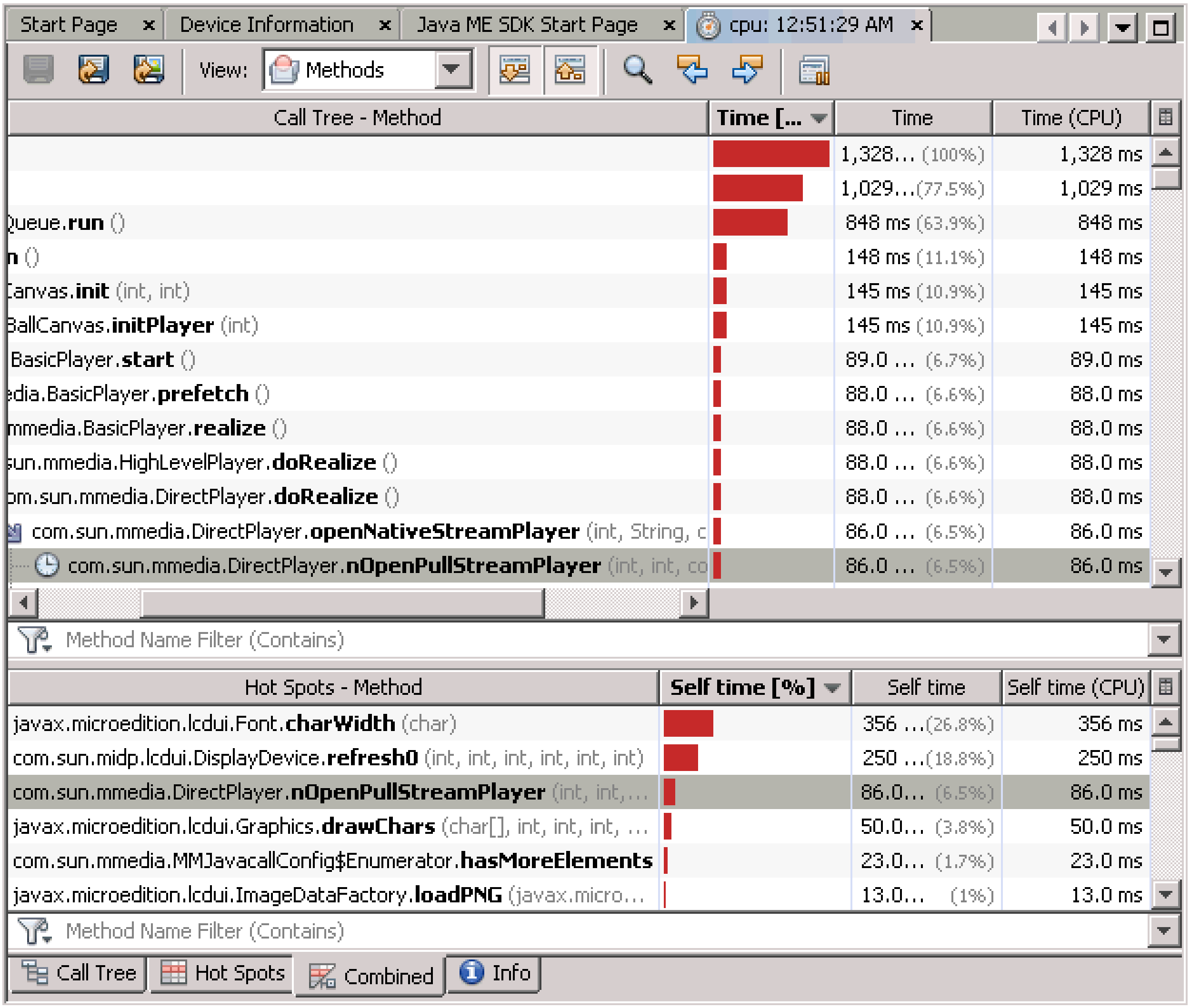
The emulator opens with your application running.

2. Interact with the application MIDlet(s) as you normally would.
3. Exit the MIDlet. You are prompted to display the saved data in a profiler:


JavaMEPhone1

There is profiling data collected.
Would you like to transfer it to a profiler?

If you choose Yes the profile data is automatically displayed in a tab labeled cpu:*time*, where *time* is the time the data was displayed.



Call Tree - Method	Time [...	Time	Time (CPU)
	🟥	1,328... (100%)	1,328 ms
	🟥	1,029...(77.5%)	1,029 ms
Queue.**run** ()	🟥	848 ms (63.9%)	848 ms
n ()	🟥	148 ms (11.1%)	148 ms
Canvas.**init** (int, int)	🟥	145 ms (10.9%)	145 ms
BallCanvas.**initPlayer** (int)	🟥	145 ms (10.9%)	145 ms
BasicPlayer.**start** ()	🟥	89.0 ... (6.7%)	89.0 ms
edia.BasicPlayer.**prefetch** ()	🟥	88.0 ... (6.6%)	88.0 ms
mmedia.BasicPlayer.**realize** ()	🟥	88.0 ... (6.6%)	88.0 ms
sun.mmedia.HighLevelPlayer.**doRealize** ()	🟥	88.0 ... (6.6%)	88.0 ms
om.sun.mmedia.DirectPlayer.**doRealize** ()	🟥	88.0 ... (6.6%)	88.0 ms
com.sun.mmedia.DirectPlayer.**openNativeStreamPlayer** (int, String, c	🟥	86.0 ... (6.5%)	86.0 ms
com.sun.mmedia.DirectPlayer.**nOpenPullStreamPlayer** (int, int, co	🟥	86.0 ... (6.5%)	86.0 ms

Method Name Filter (Contains)

Hot Spots - Method	Self time [%]	Self time	Self time (CPU)
javax.microedition.lcdui.Font.**charWidth** (char)	🟥	356 ...(26.8%)	356 ms
com.sun.midp.lcdui.DisplayDevice.**refresh0** (int, int, int, int, int, int)	🟥	250 ...(18.8%)	250 ms
com.sun.mmedia.DirectPlayer.**nOpenPullStreamPlayer** (int, int,...	🟥	86.0... (6.5%)	86.0 ms
javax.microedition.lcdui.Graphics.**drawChars** (char[], int, int, int, ...	🟥	50.0... (3.8%)	50.0 ms
com.sun.mmedia.MMJavacallConfig$Enumerator.**hasMoreElements**	🟥	23.0... (1.7%)	23.0 ms
javax.microedition.lcdui.ImageDataFactory.**loadPNG** (javax.micro...	🟥	13.0... (1%)	13.0 ms

Method Name Filter (Contains)

4. To export the profile data, press the Export icon and supply a `.nps` file name and location. This data can be reloaded at a later time. See Section 9.2, "Loading a .nps File".

5. To the save the current view to a `.png` file, press the "Save current view to image" icon and supply a file name and location.

9.2 Loading a .nps File

A previously exported `.nps` file (Section 9.1, "Collecting and Saving Profiler Data in the IDE") can be loaded at a later time.

Follow these steps to retrieve profile data:

1. Select Profile > Load Snapshot...
2. Choose the `.nps` file.

The Profiler opens in its own tab labeled cpu:*filename*. Click the Info tab at the bottom of the Method table to view the snapshot.

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

9.3 Importing a .prof File

A `.prof` file created from the command line (Section 14.3.4, "Command Line Profiling") can be loaded from the NetBeans IDE. A command line profiling session command might look like:

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

Files created from the command line are formatted differently from the `.nps` files created as described in Section 9.1, "Collecting and Saving Profiler Data in the IDE".

Follow these steps to retrieve command line profile data from the IDE:

1. Select Profile > Java ME > Import CPU Profiler Snapshot...
2. Choose the `.prof` file.

   The Profiler displays the data in its own tab labeled cpu:*filename*.

When the file has been loaded it can be saved in the `.nps` format. Click the Export to... icon and supply a file name and location.

10 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.

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

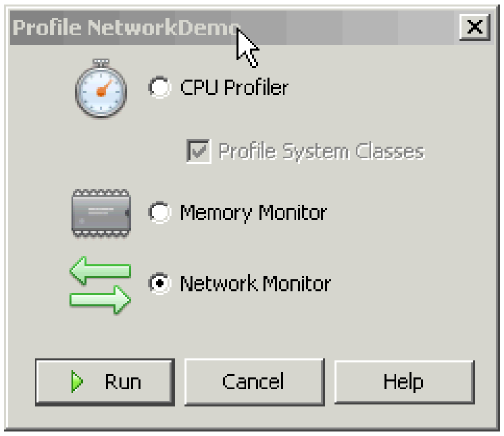
- Section 10.1, "Monitor Network Traffic"
- Section 10.2, "Filter or Sort Messages"
- Section 10.3, "Save and Load Network Monitor Information"
- Section 10.4, "Clear the Message Tree"

10.1 Monitor Network Traffic

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

1. In the Projects window right-click on a project and select Profile.
2. If this is the first time profiling this application you are prompted to integrate the profile with the project. Click Yes to perform the integration.

   In the Profile window, select Network Monitor, and click Run.



Profile NetworkDemo

( ) CPU Profiler

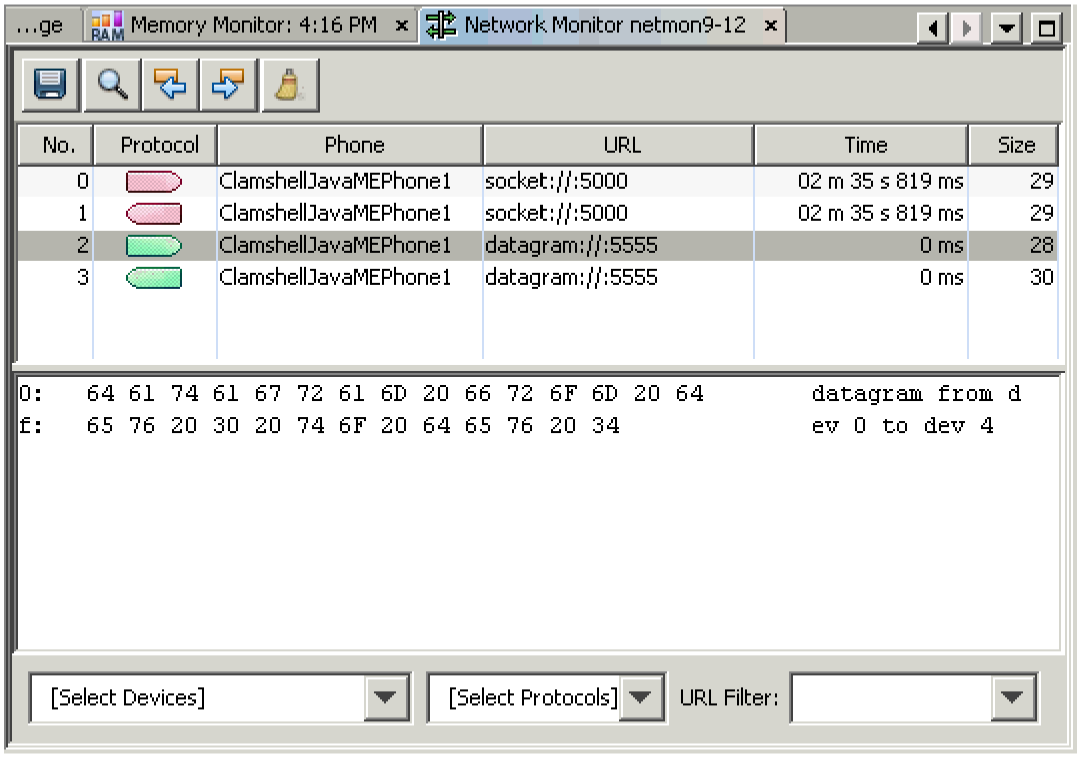
[✓] Profile System Classes

( ) Memory Monitor

(•) Network Monitor

3. 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.



No.	Protocol	Phone	URL	Time	Size
0	🟥	ClamshellJavaMEPhone1	socket://:5000	02 m 35 s 819 ms	29
1	🟥	ClamshellJavaMEPhone1	socket://:5000	02 m 35 s 819 ms	29
2	🟩	ClamshellJavaMEPhone1	datagram://:5555	0 ms	28
3	🟩	ClamshellJavaMEPhone1	datagram://:5555	0 ms	30

```
0:   64 61 74 61 67 72 61 6D 20 66 72 6F 6D 20 64        datagram from d
f:   65 76 20 30 20 74 6F 20 64 65 76 20 34               ev 0 to dev 4
```

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.

10.2 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.
- Click the magnifying glass in the Network Monitor toolbar to search for a specific string in the data in the Phone or URL columns.

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

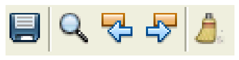
URL. Messages are sorted by URL address. Multiple messages with the same address are sorted by time.

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.

10.3 Save and Load Network Monitor Information

To save your network monitor session, click the blue disk icon at the left of the Network Monitor toolbar.



Choose a file name. The default file extension is `.nmd` (network monitor data).

To load a network monitor session, choose Profile > Java ME > Load Network Monitor Snapshot... and browse to the `.nmd` file you saved.

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

10.4 Clear the Message Tree

To remove all inactive protocol records from the network monitor choose the clear icon (the broom icon on the right of the Network Monitor tool bar).

11 Monitoring Memory

This chapter describes how to use tracing and the memory monitor to examine an application's memory use on a particular device.

Activating tracing for a particular device enables you to see low-level information as an application runs.

The Memory Monitor shows memory use as an application runs. It displays a dynamic detailed listing of the memory usage per object in table form, and a graphical representation of the memory use over time. You can take a snapshot of the memory monitor data. Snapshots can be loaded and examined later.

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

11.1 Enabling Tracing

Follow these steps to enable tracing.

1. In the Device Selector window, right-click on a device and choose Properties.
2. In the Properties window, go to the Monitor node and check the desired trace options.
   - Trace GC (garbage collection). Monitoring GC can help you determine object health. The garbage collector can't delete objects that don't have a null reference. Dead objects will be garbage collected and not reported as live.
   - Trace Class Loading. Observing class initialization and loading is useful for determining dependencies among classes.
   - Trace Exceptions. Display exceptions caught.
   - Trace Method Calls. Reports methods called and returned. The output for this option is very verbose and it can affect performance.
3. (Optional) Verbose tracing output might cause you to run out of memory on the device before the application is fully tested. You can increase the device memory as follows:

   Right-click on a device and choose Properties. From the General node, choose Heapsize, and choose a size.

Tracing data is displayed in the output window (Window > Output > Output) when an application is run on this device. It is also written to the device log, which is stored in the working directory for the device. For example:

userhome`\javame-sdk\3.2\work\JavaMEPhone1\`device.log

11.2 Using the Memory Monitor

Follow these steps to examine an application's memory use.

WARNING: Do not suspend the emulator while using the memory monitor.

---

1. In the Projects view, right-click on the project and select Profile.

   - If the profiler is not yet integrated you are prompted to enable profiling for the project. Click Yes to continue.

     The Profile window opens.

   - Select Memory Monitor, and click Run.

   The output window tab is labeled "memory-monitor" indicating that the memory monitor is active for this session. The output window displays both application status and tracing outputs for this device.

   The memory monitor opens.

2. Interact with the application as usual.

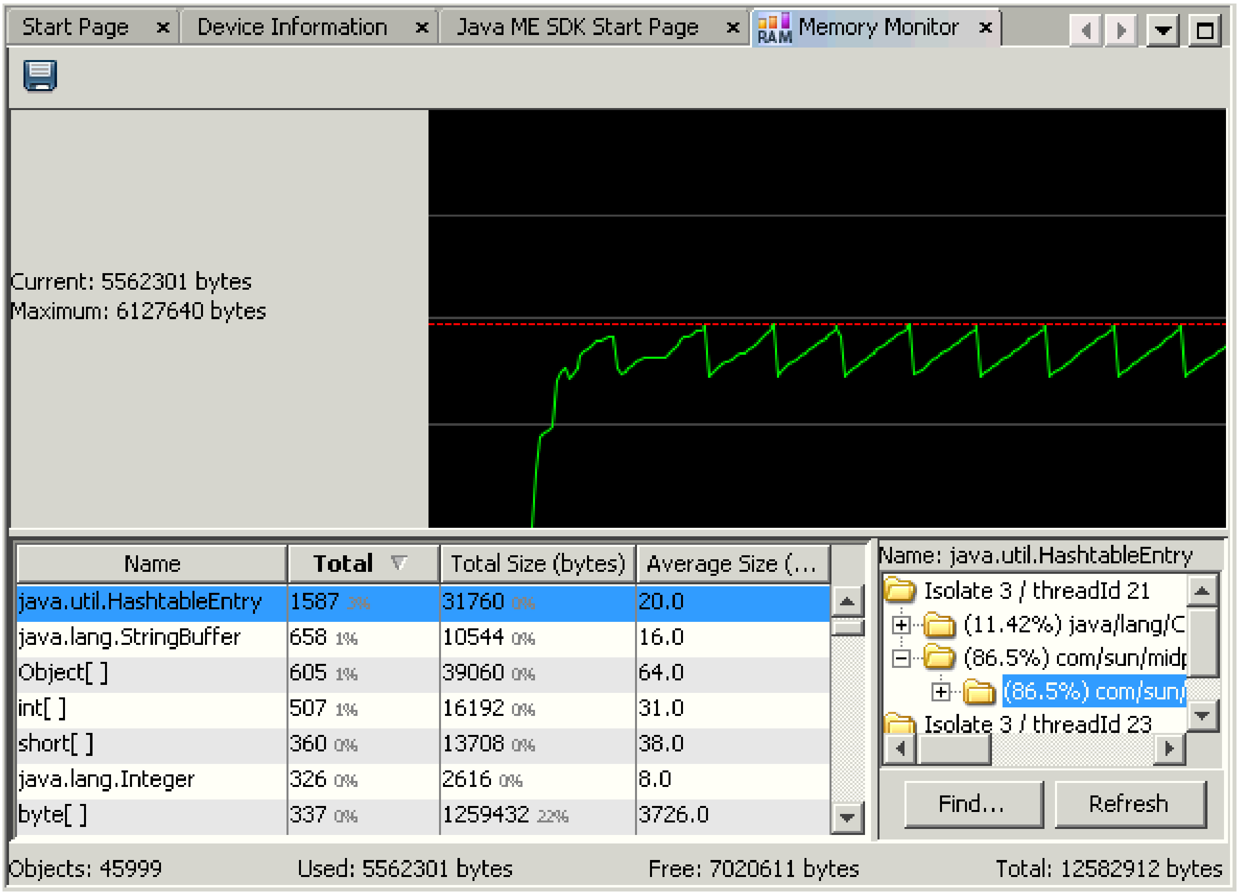
   In the Memory Monitor tab you see data displayed on the graph above and in the object table below.

   To the left of the graph you see the Current memory use in bytes. The green line plots these values. The red line is the maximum amount of memory used since program execution, corresponding to the Maximum size in bytes on the left.

   The object table columns are as follows:

   - Name. Object class name.
   - Total. Total number of objects allocated since the application began.
   - Total Size. Total amount of memory the object uses in bytes.
   - Average Size. Average object size in bytes, calculated by dividing the number of live instances by the total size.

   Beneath the table you see counters displaying the total number of objects, the amount of memory used, the amount of free memory, and the total amount of memory on the device.



Current: 5562301 bytes
Maximum: 6127640 bytes

Name	Total	Total Size (bytes)	Average Size (...
java.util.HashtableEntry	1587 3%	31760 0%	20.0
java.lang.StringBuffer	658 1%	10544 0%	16.0
Object[ ]	605 1%	39060 0%	64.0
int[ ]	507 1%	16192 0%	31.0
short[ ]	360 0%	13708 0%	38.0
java.lang.Integer	326 0%	2616 0%	8.0
byte[ ]	337 0%	1259432 22%	3726.0

Name: java.util.HashtableEntry
Isolate 3 / threadId 21
(11.42%) java/lang/C
(86.5%) com/sun/midp
(86.5%) com/sun/
Isolate 3 / threadId 23

Objects: 45999 Used: 5562301 bytes Free: 7020611 bytes Total: 12582912 bytes

3. Interact with the object table while the memory monitor is running.
   - Click a column header to sort the data. The sorting is case sensitive.
   - Click a row and the call stack tree is displayed in the window to the right of the table.
     – Double-click a folder to browse the call stacks tree to see the methods that create the object.
     – To find a particular method in the call stacks tree, click the Find button and enter a search string.
     – Click Refresh to update the call stacks tree as data is gathered. It is not refreshed automatically.
4. Take a snapshot of the memory monitor. Because the data changes rapidly it is convenient to take several snapshots and review them later.

   Click the "Save session to file" icon above the graph and specify a file name and location for the monitor data. The automatically supplied extension is `.mms`.
5. Exit the application.

   Some applications contain multiple MIDlets.
   - When you exit a MIDlet the table data is cleared.
   - The graph data is not cleared when you exit a MIDlet. The graph data you see is cumulative for this emulator session. The memory monitor plots session

data for any MIDlet run on the current emulator until you exit the application and close the emulator.

11.3 Viewing a Session Snapshot

Follow these steps to reload a memory monitor snapshot.

1. In NetBeans, select Profile >Java ME > Import Memory Monitor Snapshot...
2. Choose an `.mms` file you saved.

   The memory monitor opens in its own tab in the main window. Note the tab displays the time the snapshot was taken.

12 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.

12.1 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 a NetBeans package. 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 12.3, "Add a Different LWUIT Library".

12.2 LWUIT Resource Editor

The Resource Editor is an independent GUI tool for opening, creating, and editing resource packages for LWUIT.

To start the resource editor, go to the project view and select a project that uses the LWUIT library, then select Tools>Java ME > LWUIT Resource Editor.

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 12.1, "LWUIT and the Java ME SDK".

12.3 Add a Different LWUIT Library

The LWUIT library can be added to any CLDC/MIDP or CDC/PBP Project. A library has typically been installed as a NetBeans module.

1. Right-click on a project and select Properties.
2. In the Build category, select Libraries & Resources, and click the Add Library... button.
3. In the Add Libraries window, scroll down and select LWUIT and click Add Library.

   You can see the package under Libraries and Resources.

If you have a created a Zip or JAR from the unreleased LWUIT source you can add it in a similar fashion by pressing the Add Jar/Zip button.

12.4 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 subdemo 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.

13 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*, which is available at `http://developers.sun.com/techtopics/mobility/midp/articles/permissions/`.

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

- Section 13.1, "Security Domains"
- Section 13.2, "Setting Security Domains"
- Section 13.3, "Signing a Project"
- Section 13.4, "Managing Keystores and Key Pairs"
- Section 13.5, "Managing Root Certificates"

13.1 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.

13.2 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 13.3, "Signing a Project".

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

13.2.1 Specify the Security Domain for an Emulator

1. In the Device Selection window, right-click on the device and select Properties.
2. Find the Security Domain setting and make a selection from the context menu.

   The SDK knows the runtimes the device can support and supplies only possible domains. The default setting for the sample projects is Maximum. See Section 6.4, "Setting Device Properties".

13.2.2 Specify the Security Domain for a Project

1. Right-click on a project and select Properties.
2. In the Category area, select Running (the green triangle).
3. Select Regular Execution and check the Security domain box.

   In this context regular execution means you are running in the emulator, as opposed to running OTA.
4. Select the domain from the drop-down menu.

13.3 Signing a Project

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. The keys are as follows:

- 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 with the Keystores Manager as described in Section 13.4, "Managing Keystores and Key Pairs".

13.3.1 Sign a CLDC Project With a Key Pair

1. Right-click on a project and select Properties.
2. From the Build hierarchy, select Signing.
3. Check the Sign Distribution checkbox.
4. Choose a keystore from the Keystores drop-down menu, or click Open Keystores Manager to create a new keystore.

   The Certificate Details area displays the Alias, Subject, Issuer, and validity dates for the selected keystore.
5. Choose a key pair alias from the drop-down menu.

   A keystore might be accessed by several key pairs, each with a different alias. If you prefer to use a unique key pair, select Open Keystores Manager and create a new key pair. See Section 13.4.1.1, "Create a Keystore".
6. Click OK.

See Section 5.4.5, "Obfuscating".

13.3.2 Sign a CDC Project

To sign a CDC project use the JDK `jarsigner` command from the command line. For example:

```
jarsigner.exe -keystore keystore.ks -storepass keystorepwd MyCdcApp.jar dummyCA
```

13.4 Managing Keystores and Key Pairs

For test purposes, you can create a signing key pair to sign a MIDlet. The Keystores Manager administers this task, as described in the remainder of this topic, the key pair consists of two keys:

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

To deploy on a device, you must obtain a signing key pair from a certificate authority recognized by the device. You can also import keys from an existing Java SE platform keystore.

The following topics describe the user interface:

- Section 13.1, "Security Domains"
- Section 13.4.1.3, "Create a New Key Pair"
- Section 13.4.1.4, "Remove a Key Pair"
- Section 13.4.1.5, "Import an Existing Key Pair"

You can also use the command line tools described in Section 14.6, "Command Line Security Features".

13.4.1 Working With Keystores and Key Pairs

The Keystores Manager handles creating and using keystores. The keystores known to the Keystores Manager are listed when you sign a project, as described in Section 5.4.6, "Signing".

Keystores contain key pairs, which you can also manage from this dialog. You must select a keystore to access the key pair tools.

13.4.1.1 Create a Keystore

1. Select Tools > Keystores.

   The Keystores Manager opens.

2. Click Add Keystore...

   The Add Keystores window opens.

3. Choose Create a New Keystore.
4. Supply a name, location, and password.

   userhome`\My Documents`

5. Click OK.

   The new keystore appears in the Keystores list.

13.4.1.2 Add an Existing Keystore

1. Select Tools > Keystores.

   The Keystores Manager opens.

2. Click Add Keystore.

   The Add Keystores window opens.

3. Choose Add Existing Keystore.
4. Browse to the location of the keystore and select the keystore file. The default location for user-defined keystores is:

   userhome`\My Documents`

5. Select a keystore and Click Open, then click OK.

   The existing keystore appears in the Keystores list. You might have to unlock this keystore, and each key pair within it.

13.4.1.3 Create a New Key Pair

1. Select Tools > Keystores.

   The Keystores Manager opens.

2. Select a Keystore in the Keystores area on the left. If you prefer a different keystore, you can create one as described in Section 13.4.1.1, "Create a Keystore".

   Note, you cannot Add a key to the Built-in Keystore, but you can export a key from it.

3. Click the New... button.
4. Fill in the Create Key Pair dialog:

   You must provide an alias to refer to this key pair.

The six Certificate Details fields are provisionally optional. You must complete at least one field.

Key Pair Alias. The name used to refer to this key pair.

Common Name. Common name of a person, such as "Jane Smith"

Organization Unit. Department or division name, such as "Development"

Organization Name. Large organization name, such as "Sun Microsystems Inc."

Locality Name. Locality (city) name, such as "Santa Clara"

State Name. State or province name, such as "California"

Country. Two-letter country code, such as "US"

You must provide a password at least six characters long.

5. Click OK.

   The new key is displayed in the Keys area under its alias. You can now select this keypair when you sign a project. See Section 13.3, "Signing a Project".

13.4.1.4 Remove a Key Pair

1. Select Tools > Keystores.
2. In the Keys area, click a Key Pair.
3. Select Delete. You are asked if you are sure. Click Yes if you are and the delete proceeds.

13.4.1.5 Import an Existing Key Pair

If you have keys in a Java SE platform keystore that you would like to use for MIDlet signing, you can import the signing keys to the Java ME SDK.

1. Select Tools > Keystores.
2. Click Add Keystore...

   The Add Keystore window opens.
3. Click Add Existing Keystore.
4. Browse to the keystore location.
5. Click OK.

13.5 Managing Root Certificates

The Oracle Java ME SDK command line tools described in Section 14.6.3, "Manage Certificates (MEKeyTool)" manage the emulator's list of root certificates.

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.

Each emulator instance has its own `_main.ks` file located in its `appdb` directory. For example: *userhome*`\javame-sdk\3.2\work\`*devicename*`\appdb`.

You can use the `-import` option to import certificates from these keystores as described in Section 14.6.3, "Manage Certificates (MEKeyTool)".

14 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.

14.1 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://java.sun.com/j2me/docs/uei_specs.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.

14.2 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 14–1 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
```

14.3 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 14–2 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 14.3.1, "MIDlet Options", Section 14.3.2, "CDC Options", and Section 14.3.3, "Debugging and Tracing Options".

14.3.1 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://java.sun.com/products/javadevice/overview.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 runover-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 14.3, "Emulator Command Line Options" and Section 14.3.3, "Debugging and Tracing Options".

14.3.2 CDC Options

The following options apply to CDC projects.

- `-Xmain:`*main-class-name*

  Run the main method of a Java class, as in Java SE.

- `-Xxlet:classpath=`*class-path*, `class=`*fully-qualified-name*, `[arg=`*argument*`]*`

  Run an Xlet application.

See also Section 14.3, "Emulator Command Line Options" and Section 14.3.3, "Debugging and Tracing Options".

14.3.3 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 graphical debugger from NetBeans, select Debug > Attach 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`

14.3.4 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\NetBeansProjects\Games\d
ist\Games.jad" -Xprofile:file=C:\temp\Games.prof
```

When you launch the emulator and profile an application from the command line the data profile you save has a different format than `.nps` files created with the Profile option in the NetBeans IDE.

Files created from the command line should be given the extension `.prof` to distinguish them from IDE profiler files.

To view `.prof` files in the IDE, select Profile > Java ME > Import CPU Profiler Snapshot... Your file is displayed in a tab labeled with the name of the fiile containing the snapshot.

When the file is loaded in the IDE you can export the data in `.nps` form, using the Export to... feature as described in Section 9.1, "Collecting and Saving Profiler Data in the IDE", step 5. These files can be loaded using Profile > Java ME > Import CPU Profiler Snapshot...

14.4 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 14.4.1, "Check Prerequisites"
- Section 14.4.2, "Compile Class Files"
- Section 14.4.3, "Preverify Class Files"

14.4.1 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.

14.4.2 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.

14.4.3 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.

14.5 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
```

14.6 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 14.6.1, "Change the Default Protection Domain"
- Section 14.6.2, "Sign MIDlet Suites (jadtool)"
- Section 14.6.3, "Manage Certificates (MEKeyTool)"

14.6.1 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 13.1, "Security Domains". For example, `-Xdomain:maximum`.

14.6.2 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. `jadtool` is also capable of signing payment update (JPP) files.

`jadtool` only uses certificates and keys from Java SE platform keystores. Java SE software provides `keytool`, the command-line tool to manage Java SE platform keystores.

`jadtool` is packaged in a JAR file. To run it, open a command prompt, change the current directory to *installdir*`\bin`, and enter the following command:

`jadtool` *`command`*

The commands are as follows:

- `-help`

  Prints the usage instructions for `jadtool`.

- `-addcert -alias` *alias* `[-keystore` *keystore*`] [-storepass` *password*`] [-storetype` *PKCS11*`] [-certnum` *number*`] [-chainnum` *number*`] [-encoding` *encoding*`] -inputjad | inputjpp` *input-file* `-outputjad | outputjpp` *output-file*

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

- `[-jarfile <`*filename*`>] -keypass <`*password*`> -alias <`*key alias*`> -storepass <`*password*`> [-keystore <none|keystore>] [-storetype <PKCS11>] [-encoding <`*encoding*`>] -inputjad <`*filename*`> -outputjad <`*filename*`>`

  Adds the digital signature of the given JAR file to the specified JAD file. The default value for `-jarfile` is the `MIDlet-Jar-URL` property in the JAD file.

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

  Displays information about certificates in JAD and JPP files.

- `-addjppsig -keypass <`*password*`> -alias <`*key alias*`> [-storepass <`*password*`>] [-keystore <none|keystore>] [-storetype <`*PKCS11*`>] [-encoding <`*encoding*`>] -inputjpp <`*filename*`> -outputjpp <`*filename*`>`

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

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`

14.6.3 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.

14.7 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 14–3 lists the `wscompile` options:

Table 14–3 *`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

Table 14–3 (Cont.) `wscompile` ***Options***

Option	Description
`-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 14–4 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 14–4 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
`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
```

15 Logs

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

15.1 Device Manager Logs

The device manager log is placed into:

userhome`\javame-sdk\`*version*`\log\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>
```

15.2 Device Instance Logs

Each device (or emulator) instance writes its own log into its directory. See Table 8–1 to correlate the emulator number and the device name.

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

`log4j.xml` controls the verbosity of the device instance logs, as described in Section 15.1, "Device Manager Logs".

16 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 16–1 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 16.2, "Oracle APIs" describes Oracle APIs provided to support the IMP-NG platform.

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

16.1 JCP APIs

Table 16–1 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 16–1 (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 209, AGUI 1.0	*Advanced Graphics and User Interface Optional Package for the J2ME Platform* http://www.jcp.org/en/jsr/detail?id=209
JSR 211, CHAPI	*Content Handler API* http://jcp.org/en/jsr/detail?id=211
JSR 217, PBP 1.1	*Personal Basis Profile 1.1* http://www.jcp.org/en/jsr/detail?id=217
JSR 218, CDC 1.1	*Connected Device Configuration 1.1* http://jcp.org/en/jsr/detail?id=218
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

16.2 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 Javadocs for these APIs are in your installation's \docs directory.

17 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.

17.1 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.

17.2 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.

17.3 Running PDAPDemo

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

17.3.1 Browsing Files

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

userhome`\javame-sdk\`*version*`\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.

17.3.2 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.

18 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 18.2, "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 18.3, "Running the OBEX Demo".

18.1 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 right-click on a CLDC device and select Properties.

   The device properties are displayed in the Properties window. If you do not see this window, select Window > Properties from the NetBeans toolbar.

2. Scroll down to see the Bluetooth and OBEX properties. When you click a property a description is shown in the description area. If you can not see this area, right click a property and select Show 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.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.

18.2 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 softbutton 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.

18.3 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.

19

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

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

19.1 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)

MIME Type	Description
video/3gpp*	Third generation mobile broadband with video
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.

19.1.1 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.

19.2 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.

19.3 Ring Tones

MMAPI plays ring tones, as demonstrated in Section 19.5.1, "Simple Tones" and Section 19.5.2, "Simple Player". The ring tone formats mentioned here are in common use. You can download ring tones or create your own.

19.3.1 Download Ring Tones

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

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

19.3.2 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#****
  ```

19.4 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.

19.5 Running MMAPIDemos

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

19.5.1 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.

19.5.2 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 ringtone file (.jts format) is stored in the MIDlet suite JAR file.
- WAV Music plays a brief audio file. The display shows the title of the audio file, the duration of the audio the elapsed time during playback, and the playback rate in percent. This WAV file is retrieved from an HTTP server.
- MIDI Scale plays a MIDI file that is retrieved from an HTTP server.
- The Animated GIF example shows an animated GIF that counts from 1 to 5. The file is stored in the MIDlet suite JAR file.
- 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.

19.5.3 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 as a Form Item. The form also includes some information about the playback, including the current time. Choose the Snapshot command to take a snapshot of the running animation. The snapshot is placed in the form following the animated GIF.

- **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.

19.5.4 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.

20 JSR 172: Web Services Support

The Oracle Java ME SDK emulator supports JSR 172, the J2ME Web Services Specification. JSR 172 provides APIs for accessing web services from mobile applications. It also includes an API for parsing XML documents.

20.1 Generating Stub Files from WSDL Descriptors

The NetBeans IDE provides a stub generator that automates creating source code for accessing web services that conform to the J2ME Web Services Specification. You can add stubs to any MIDP application.

Note: If you are using NetBeans 7.1.2, or 7.2.1 or higher the "SOAP Web Services" plugin must be installed and activated.

The following is a general procedure for adding stubs:

1. In the Projects window, expand the tree for a project.
2. Launch the Java ME Web Service Client wizard:

   Right-click on the Source Packages node and select New > Java ME Web Service Client...
3. In the New Java ME Webservice Client page, you can either:
   - Click "Running Web Service and enter the URL for the WSDL", then click Retrieve WSDL.

     Or,
   - Click "Specify the Local filename for the retrieved WSDL" and browse to a file on your system.

   In either case, you must enter a Package name (if it is not supplied), then click Finish. The new package appears in the project and includes an interface file and a stub file.
4. You can now edit your source files to call the content the stub provides, then build and run.

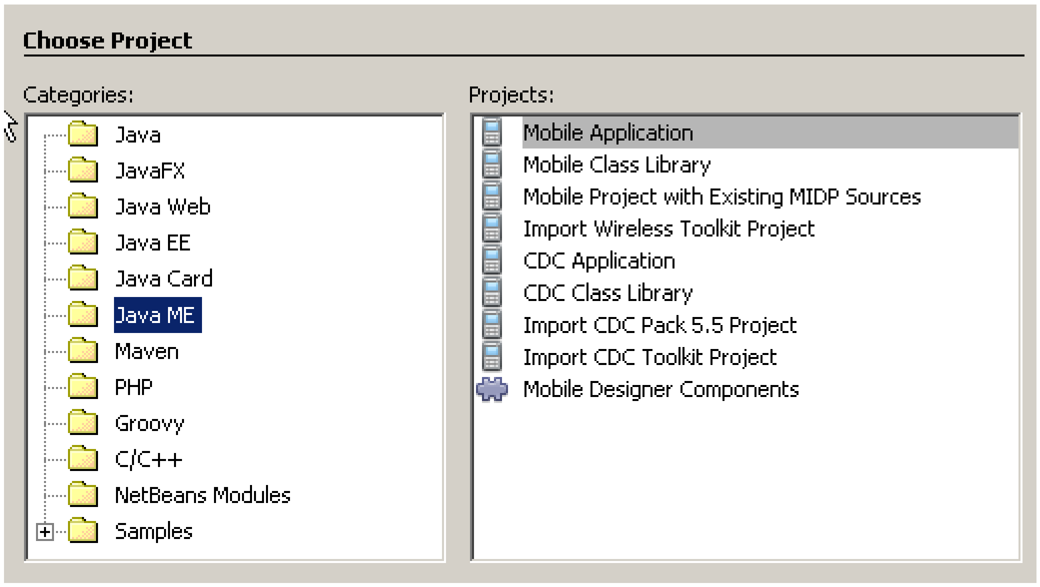
   See Section 20.2, "Creating a New Mobile Web Service Client" for a step by step process, or see Section 20.3, "Run JSR172Demo" and view the demo source files.

20.2 Creating a New Mobile Web Service Client

This sample procedure creates a new project and adds a web service client. However, you can add a web service client to any MIDP project, it does not have to be new.

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".

1. Select File > New Project, choose the Java ME category, and the Mobile Application project type, and click Next.

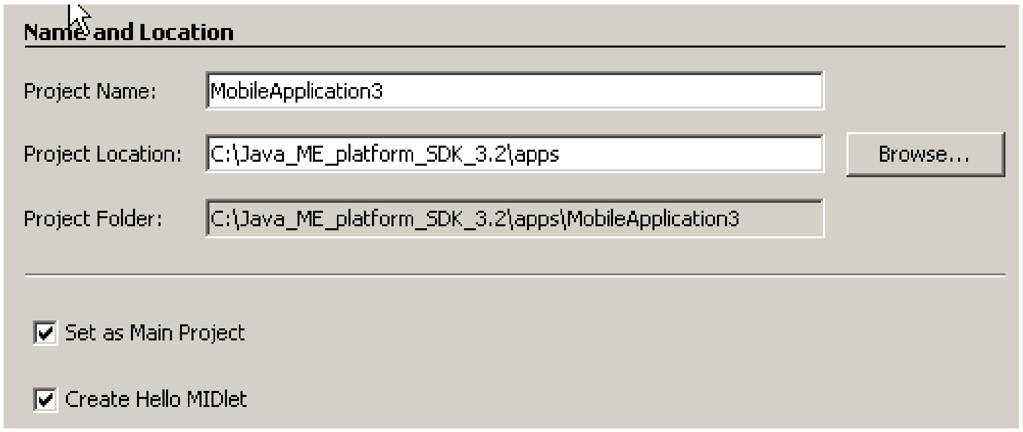


Choose Project

Categories: Java ME

Projects: Mobile Application

2. Name your project and ensure Create Hello MIDlet is checked. Click Next.



Name and Location

Project Name: MobileApplication3

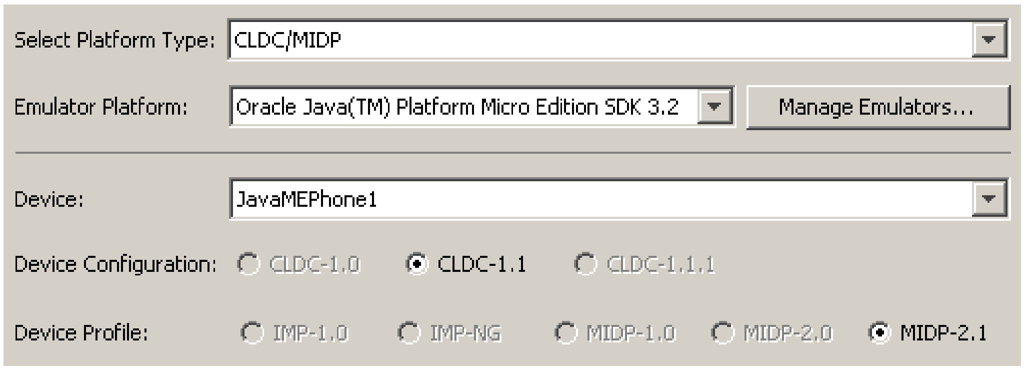
Project Location: C:\Java_ME_platform_SDK_3.2\apps

Project Folder: C:\Java_ME_platform_SDK_3.2\apps\MobileApplication3

☑ Set as Main Project

☑ Create Hello MIDlet

3. Ensure that SDK 3.2 is the platform and choose a device. Click Next.



Select Platform Type: CLDC/MIDP

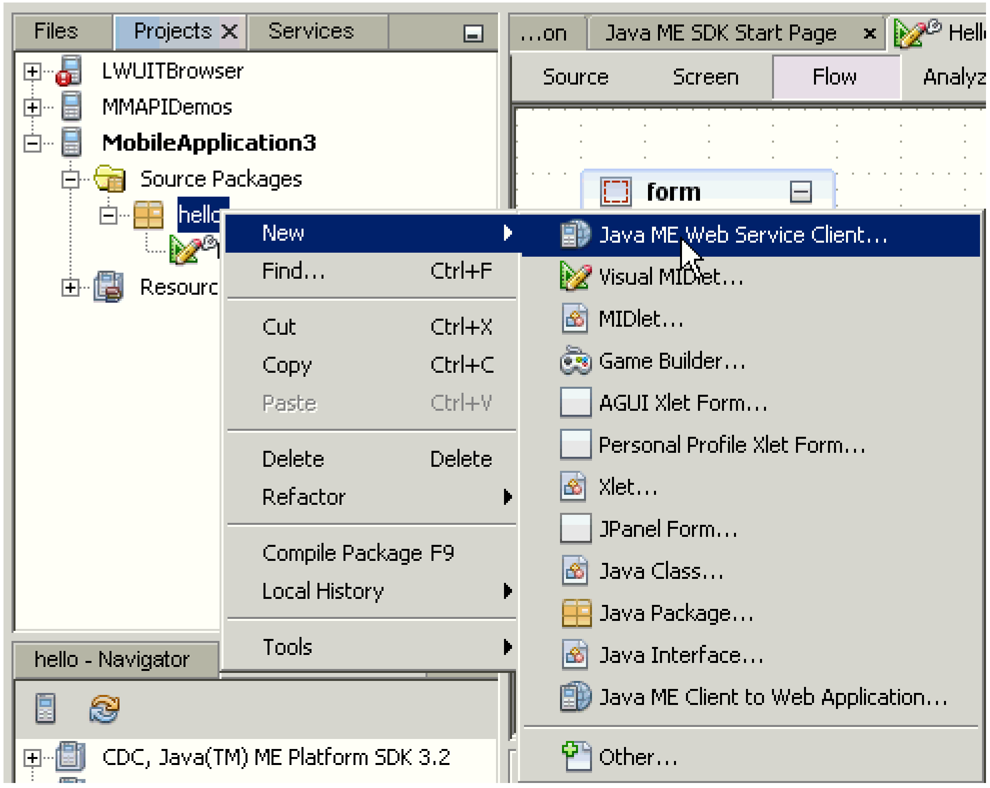
Emulator Platform: Oracle Java(TM) Platform Micro Edition SDK 3.2 Manage Emulators...

Device: JavaMEPhone1

Device Configuration: CLDC-1.0 [✓ CLDC-1.1] CLDC-1.1.1

Device Profile: IMP-1.0 IMP-NG MIDP-1.0 MIDP-2.0 [✓ MIDP-2.1]

4. Right-click on the Source Packages node and select New > Java ME Web Service Client...



5. In the New Java ME Webservice Client page:

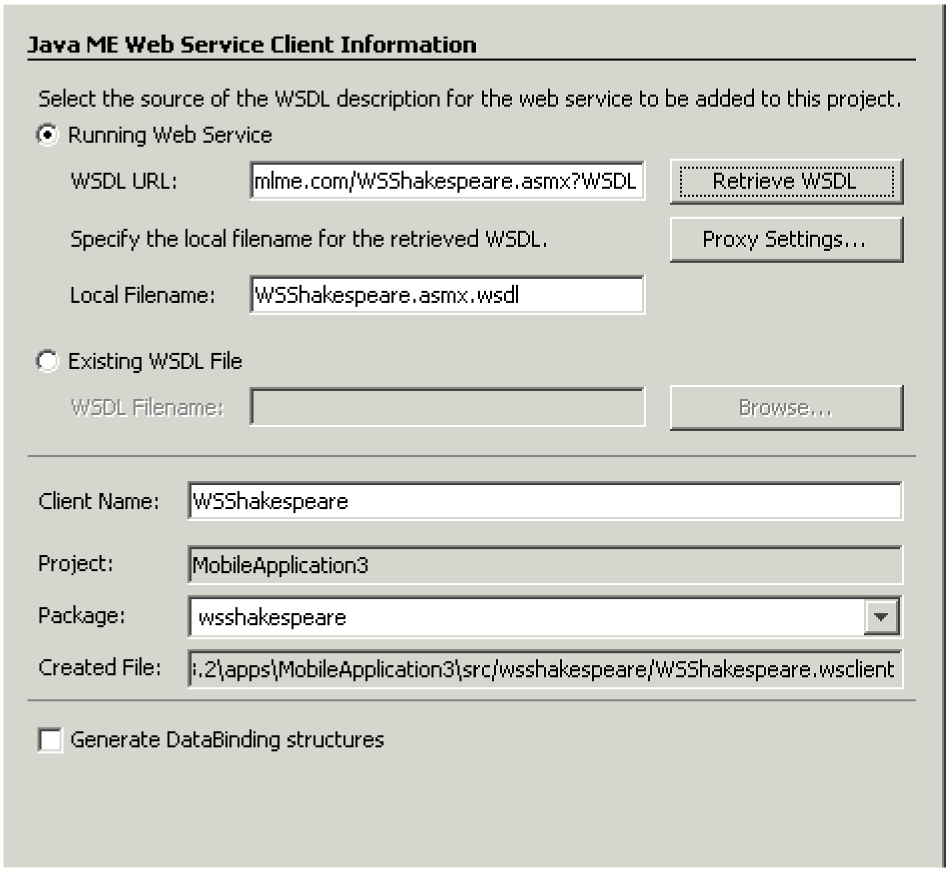
- Click Running Web Service and in the WSDL URL field, enter:

  `http://www.xmlme.com/WSShakespeare.asmx?WSDL`

  Click Retrieve WSDL.

- The Package name is wsshakespeare.

Click Finish.


Java ME Web Service Client Information

Select the source of the WSDL description for the web service to be added to this project.

[● Running Web Service]

WSDL URL: mlme.com/WSShakespeare.asmx?WSDL

Specify the local filename for the retrieved WSDL.

Local Filename: WSShakespeare.asmx.wsdl

○ Existing WSDL File

WSDL Filename:

Client Name: WSShakespeare

Project: MobileApplication3

Package: wsshakespeare

Created File: :.2\apps\MobileApplication3\src/wsshakespeare/WSShakespeare.wsclient

☐ Generate DataBinding structures

http://www.xmlme.com/WSShakespeare.asmx?WSDL

****

The new package appears in the Source Packages tree and includes `Shakespeare.java` and `Shakespeare_Stub.java`.

6. Edit `HelloMIDlet.java` as follows:

   - At the beginning, replace the default import declarations with:

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

   - Locate the `startApp()` method and replace its contents with the following code:

```
String text;
Shakespeare s = new Shakespeare_Stub();
 try {
     text = s.getSpeech("Romeo");
 } catch (java.rmi.RemoteException rex) {
     text = "error";
     System.out.println(rex.getMessage());
 }
 TextBox t = new TextBox("Hello", text, 2048, 0);

 final Command exitCommand = new Command("Exit", Command.EXIT, 1);
 t.addCommand(exitCommand);
 t.setCommandListener(new CommandListener() {
```

```
            public void commandAction(Command c, Displayable d) {
              if (c == exitCommand) {
           notifyDestroyed();
              }
            }
        });

        Display.getDisplay(this).setCurrent(t);
```

7. Build and run the project. You see a quote from Shakespeare's Romeo and Juliet on the device screen.

   You can vary the above procedure to use a local WSDL file. Open the following web page in a browser:

   `http://www.xmlme.com/WSShakespeare.asmx?WSDL`

   Save it to a local file. For example, `C:\ws\WSShakespeare.wsdl`. Follow the procedure above, except at Step 4, specify the local file name.

20.3 Run JSR172Demo

JSR172Demo shows how to access a web service from a MIDlet. The web service in this demo is running on an Internet server, and it conforms to the J2ME Web Services Specification. The client is the MIDlet running in the emulator

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".

Set Up the GlassFish Server

This demo requires the Oracle GlassFish server.

- If you installed a full version of NetBeans you probably have a GlassFish installation. Choose Tools > Servers and choose Glassfish to view the defaults.
- If you do not have Glassfish, it can be downloaded from:

  http://www.oracle.com/technetwork/java/javaee/downloads

Set Up Environment Variables

Set the following environment variables:

`JAVA_HOME=`*Java-installation-path*

`GLASSFISH_HOME=`*GlassFish-installation-path*

Run the Demo Scripts

The scripts in the `\server` subdirectory assume that your server has a domain named "domain1" in which the service is automatically deployed. If you do not have a domain named domain1, set up this domain or edit `run.bat` and specify a domain you already have. Run the scripts:

demo_directory`\JSR172Demo\server\build.bat`

demo_directory`\JSR172Demo\server\run.bat`

Verify the Web Service

- Start the Glassfish service.
- In a browser, open the following URL:

  `http://localhost:8080/serverscript/serverscript`

You see a page titled Web Services that displays information about the service including links to the WSDL file corresponding to the localhost url.

Run the MIDlet in the Emulator

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

21 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.

21.1 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
```

21.2 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 21.4.4, "MohairMIDlet".

21.3 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.

21.3.1 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
   }
```

21.3.2 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.

21.3.3 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
    }
  }
}
```

21.4 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 21.2, "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.

For each SATSA demo, start with this sequence:

1. Go to File > New Project and in the Categories window select Samples > Java ME SDK 3.2 and single-click SATSADemos. Click Next. Save the sample project in:

   installdir`\apps\SATSADemos`

   Click Finish.

2. Right-click on the project, select Properties and choose Running. Enable Regular execution and check Specify the Security Domain. Choose maximum from the list.
3. Start the instance(s) of `cref` from the command line.
4. Run the project.

21.4.1 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 21.2, "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. Right-click on the project, select Properties, and choose Running. Enable Regular execution and check Specify the Security Domain. Choose maximum from the list.
2. 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
   ```

3. 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.

21.4.2 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.

21.4.3 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.

21.4.4 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.

21.4.5 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. 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 Running. Enable Regular execution and check Specify the Security Domain. Choose maximum from the list.

2. Start `cref` from the SDK installation directory as follows:

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

3. 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.

22 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.

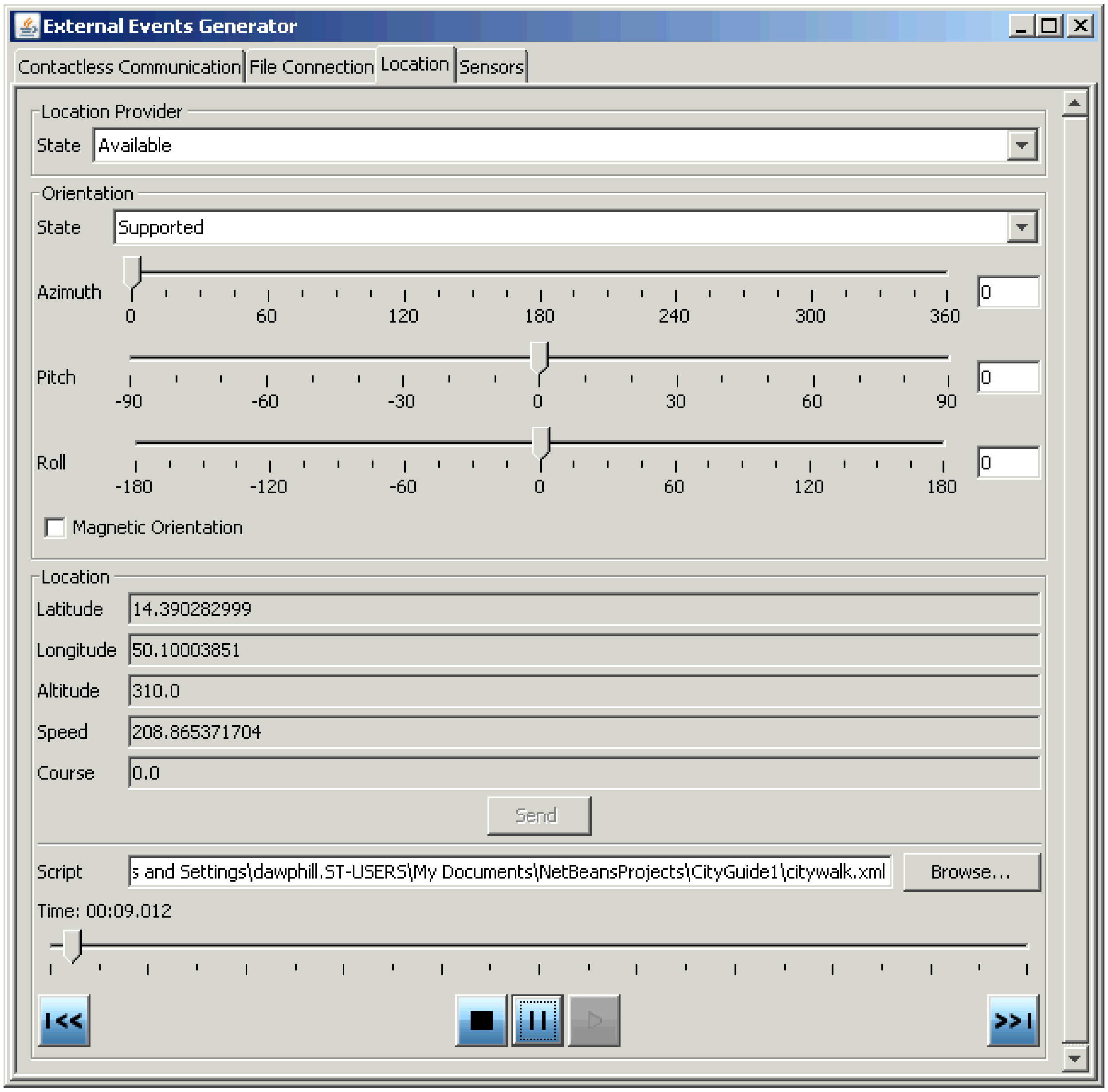
22.1 Setting the Emulator's Location at Runtime

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>
```



State: Available

State: Supported

Azimuth: 0 (0, 60, 120, 180, 240, 300, 360)

Pitch: 0 (-90, -60, -30, 0, 30, 60, 90)

Roll: 0 (-180, -120, -60, 0, 60, 120, 180)

Latitude: 14.390282999

Longitude: 50.10003851

Altitude: 310.0

Speed: 208.865371704

Course: 0.0

Script: s and Settings\dawphill.ST-USERS\My Documents\NetBeansProjects\CityGuide1\citywalk.xml

Time: 00:09.012

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.

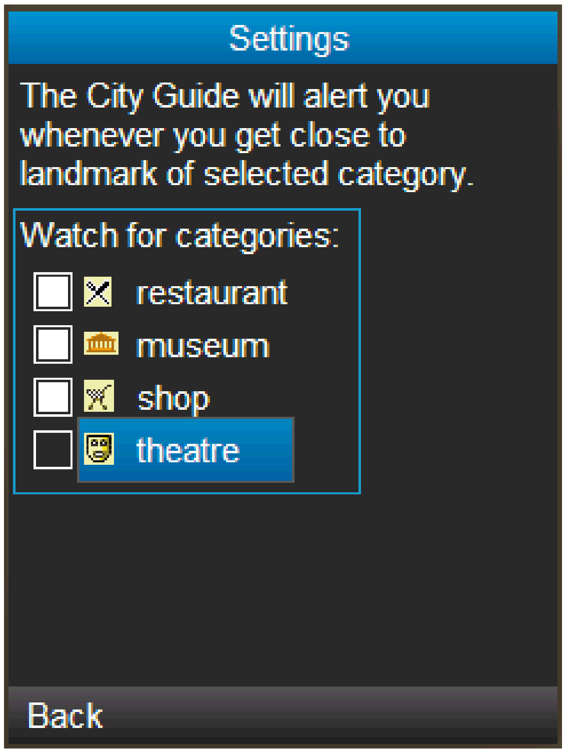
22.2 Running the CityGuide Sample Project

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 13.1, "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

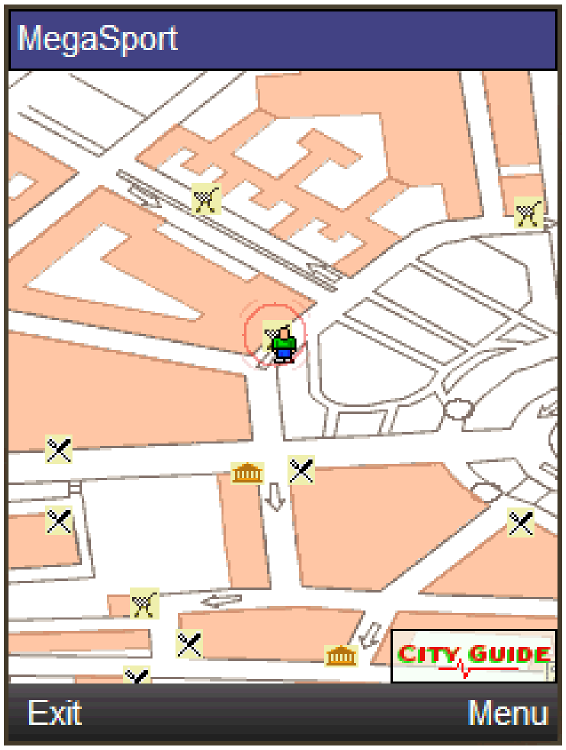
The City Guide will alert you whenever you get close to landmark of selected category.

Watch for categories:

- [ ] restaurant
- [ ] museum
- [ ] shop
- [ ] theatre

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

23 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 23.3, "Running WMA Tool".

23.1 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.

See Section 23.1.2, "WMA Console Interface" or Section 23.3, "Running WMA Tool".

23.1.1 Launching the WMA Console

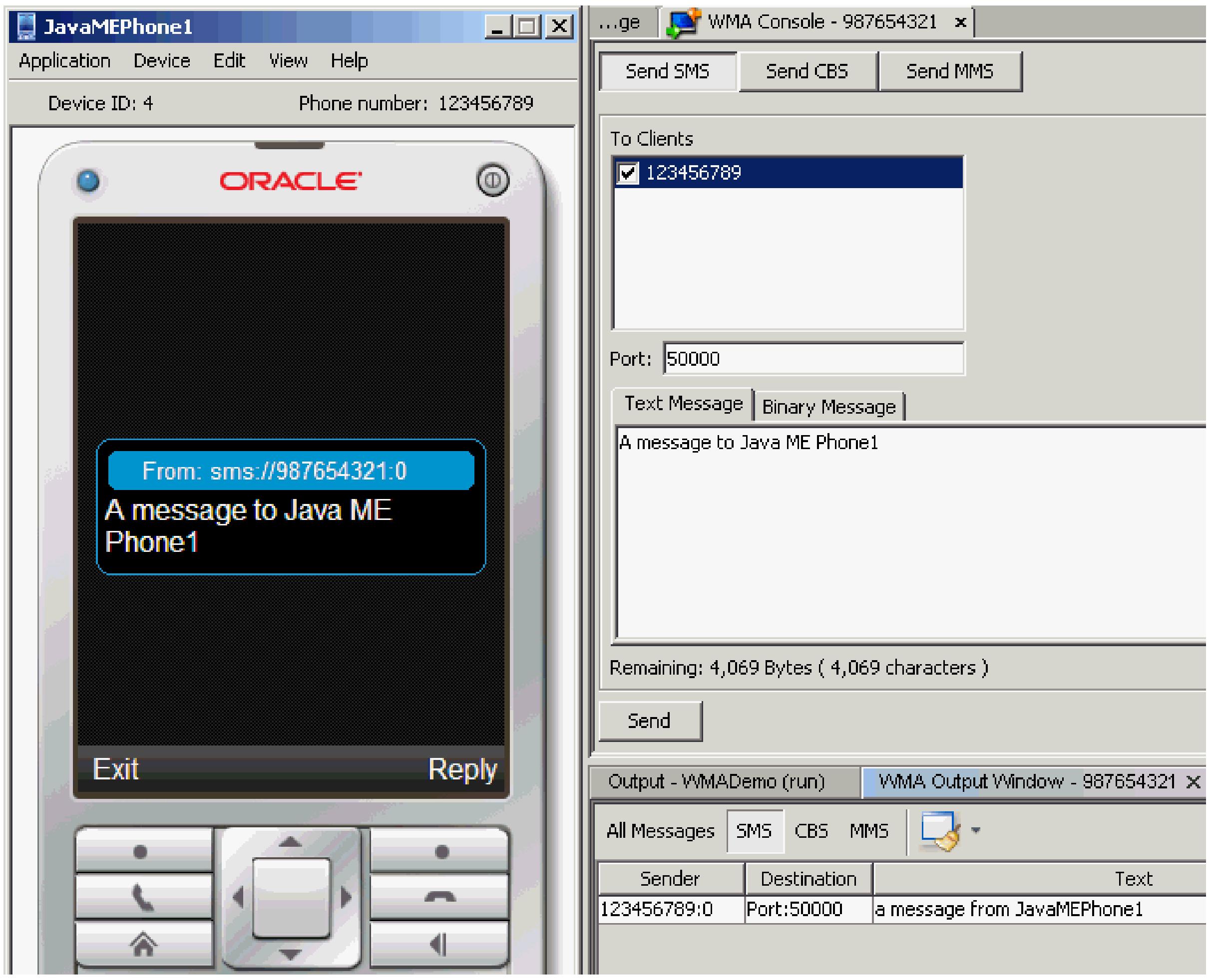
To launch the WMA console, select Tools > Java ME > WMA Console. Messages can be sent from the WMA Console to an emulator instance.

The console opens as a tab in the NetBeans 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. In the Device Selector, right-click on the CLDC, Java(TM) ME Platform SDK 3.2 node in the device selector, and select Properties. Type a new value in the WMA Console Phone Number 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 NetBeans IDE.

23.1.2 WMA Console Interface

To open the WMA Output window, select Window > Output > WMA Console Output. This window displays messages received from an emulator. By default it is docked at the bottom of the NetBeans IDE.



Device ID: 4 Phone number: 123456789

From: sms://987654321:0
A message to Java ME Phone1

To Clients
[✓] 123456789

Port: 50000

A message to Java ME Phone1

Remaining: 4,069 Bytes (4,069 characters)

Sender	Destination	Text
123456789:0	Port:50000	a message from JavaMEPhone1

23.1.3 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.

23.1.4 Sending a Text or Binary SMS Message

To launch the WMA console, select Tools > Java ME > WMA Console. To open the WMA Output window, select Window > Output > WMA Console Output.

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 23.2.1, "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 23.2.3, "Sending SMS Messages From WMA Console to an Emulator and Back".

23.1.5 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 23.2.1, "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.

To try this yourself see Section 23.2.4, "Sending CBS Messages from WMA Console to an Emulator".

23.1.6 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 23.2.1, "WMADemo Push Registry Values").

  To remove a part, select it and press Remove.

To try this yourself, see Section 23.2.5, "Sending MMS Messages from WMA Console to an Emulator".

23.1.7 Receiving Messages in the WMA Console

To start the WMA console, select Tools > Java ME > 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.

23.2 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.

23.2.1 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, select the Application Description category and view the Push Registry tab.

- 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`.

23.2.2 Running WMADemo OTA

Because this sample uses the push registry, you cannot see all of its features with the regular execution process. You must install the application into the emulator using the over the air provisioning capability that mirrors how applications are installed on real devices.

1. Right-click the WMADemo project and select Properties from the context menu.
2. Select the Running Category and choose the Execute through OTA radio button. Click OK.
3. Run WMADemo in an emulator.

   Wait a few seconds for the application to download to the emulator and register itself.

   The application home screen shows the MIDlets you can launch: SMS Send, SMS Receive, CBS Receive, MMS Send and MMS Receive.
4. Launch the WMA console (see Section 23.1.1, "Launching the WMA Console").

23.2.3 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 Tools > Java ME > WMA Console. To open the WMA Output window, select Window > Output > WMA Console Output. The WMADemo should be running in the emulator, as described in Section 23.2.2, "Running WMADemo OTA".
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.)

23.2.4 Sending CBS Messages from WMA Console to an Emulator

This process is similar to sending SMS Messages. Instead of specifying a port number you specify a Message Identifier.

1. To launch the WMA console, select Tools > Java ME > WMA Console. To open the WMA Output window, select Window > Output > WMA Console Output.
2. Click on the Send button in the WMA console window.

   Supply a Message Identifier of 50001.

   Type your text message or attach a binary message and click Send.

3. The emulator asks if it is OK if the `WMADemo` interrupts and if it can be launched. You might receive several permission requests based on your firewall settings.

   Choose Yes. The `CBSReceive` MIDlet is launched and immediately displays the incoming message. Click Exit to close the MIDlet.

23.2.5 Sending MMS Messages from WMA Console to an Emulator

To send an MMS message from the WMA console to the emulator, ensure that `WMADemo` has been installed using Run Project via OTA.

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. If you do not see the number you want, click the Add button to add it. When you have listed multiple numbers the number field is a dropdown list

   The To field on the left is a dropdown list from which you can choose To, Cc or Bcc.

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.

23.3 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.

23.3.1 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]
```

Example

This example demonstrates how to receive a message from an emulator.

1. Start the emulator from the Windows Start menu:

   Start > Programs > Java(TM) ME Platform SDK 3.2 > Java ME SDK CLDC Emulator.

   You can also start the emulator from the bin directory. This example also runs the WMADemo project.

```
emulator.exe -Xdevice:JavaMEPhone1
-Xdescriptor:"C:\Documents and Settings\user\My Documents\NetBeansProje
cts\WMADemo\dist\WMADemo.jad"
```

2. Start wma-tool from the Java ME SDK *installdir*`\bin` directory:

```
C:\Java_ME_platform_SDK_3.2\bin\wma-tool smsreceive

WMA tool started with phone number: 987654321
press <Enter> to exit.
```

3. In the emulator run the SMS Send MIDlet and send a message to the WMA console. Enter the console telephone number

   The console receives the message as follows:

```
SMS Received:
          From: 123456789
Timestamp: Thu Aug 23 23:31:26 PDT 2012
          Port: 50000
  Content type: Text
      Encoding: GSM7BIT
       Content: A message from JavaMEPhone1 to wma-tool
Waiting for another message, press <Enter> to exit.
```

23.3.2 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"
  ```

23.3.3 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"
```

23.3.4 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
```

24 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`.

24.1 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.

24.1.1 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.

24.1.2 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.

24.2 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.

24.3 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.

24.4 Running Demo3D Samples

Demo3D contains MIDlets that demonstrate JSR 184 features.

Go to File > New Project and in the Categories window select Samples > Java ME SDK 3.2 and single-click Demo3D and Click Next. Specify a name and location and click Finish.

24.4.1 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 24–1 provide control over the game.

Table 24–1 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 NetBeans projects.

24.4.2 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`

24.4.3 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`

25

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.

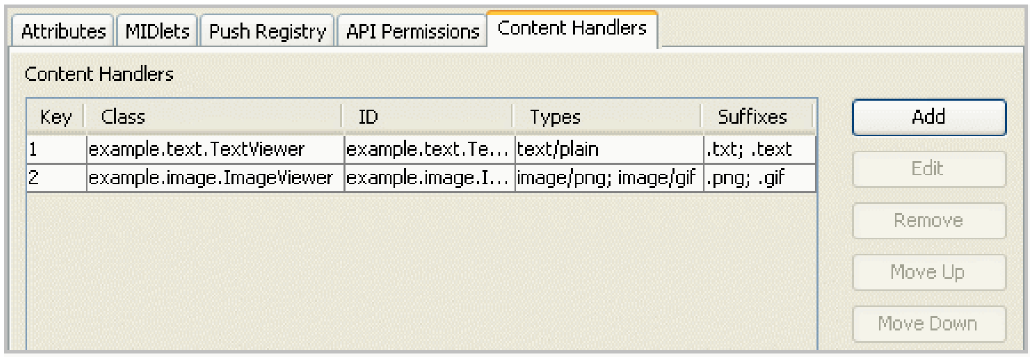
See Section 25.1, "Using Content Handlers" and Section 25.4, "Running the CHAPIDemo Content Browser".

25.1 Using Content Handlers

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

Follow these steps to work with content handlers in the CHAPIDEMO sample application (see Section 25.4, "Running the CHAPIDemo Content Browser").

1. In the Projects window, right-click CHAPIDemo and choose Properties from the context menu.
2. In the Category pane, select Application Descriptor, and click the Content Handlers tab.
3. In the Content Handlers table, each line in the list represents the settings for a content handler.



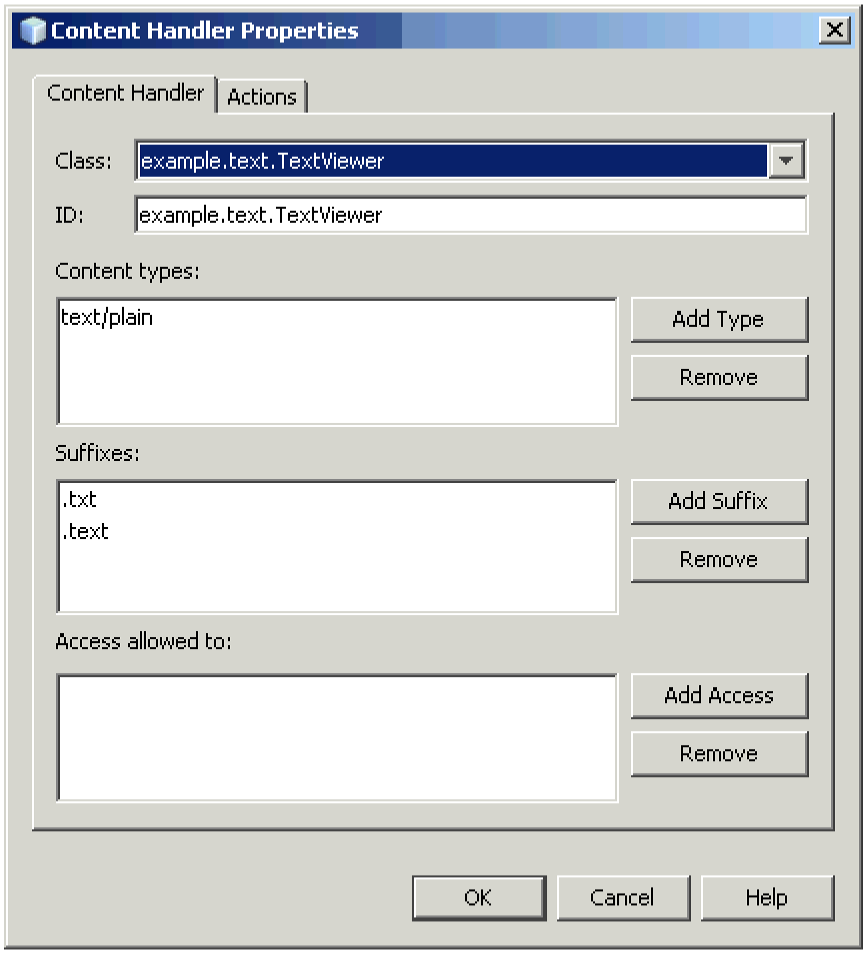
Key	Class	ID	Types	Suffixes
1	example.text.TextViewer	example.text.Te...	text/plain	.txt; .text
2	example.image.ImageViewer	example.image.I...	image/png; image/gif	.png; .gif

- To create a new content handler, press Add, or to edit an existing content handler, press Edit. Both actions open the Content Handler Properties window. See Section 25.2, "Defining Content Handler Properties".

- To adjust the order of the content handlers, select one and using the Move Up and Move Down buttons. To remove a content handler from the list, select it and press Remove.
- See Section 25.2, "Defining Content Handler Properties" and Section 25.4, "Running the CHAPIDemo Content Browser"

25.2 Defining Content Handler Properties

In the Projects window, right-click on a project and choose Properties from the context menu. In the Category pane, select Application Descriptor, and click the Content Handler tab. Pressing Add or Edit opens the Content Handler Properties window.



Class: example.text.TextViewer

ID: example.text.TextViewer

Content types:

text/plain

Suffixes:

.txt
.text

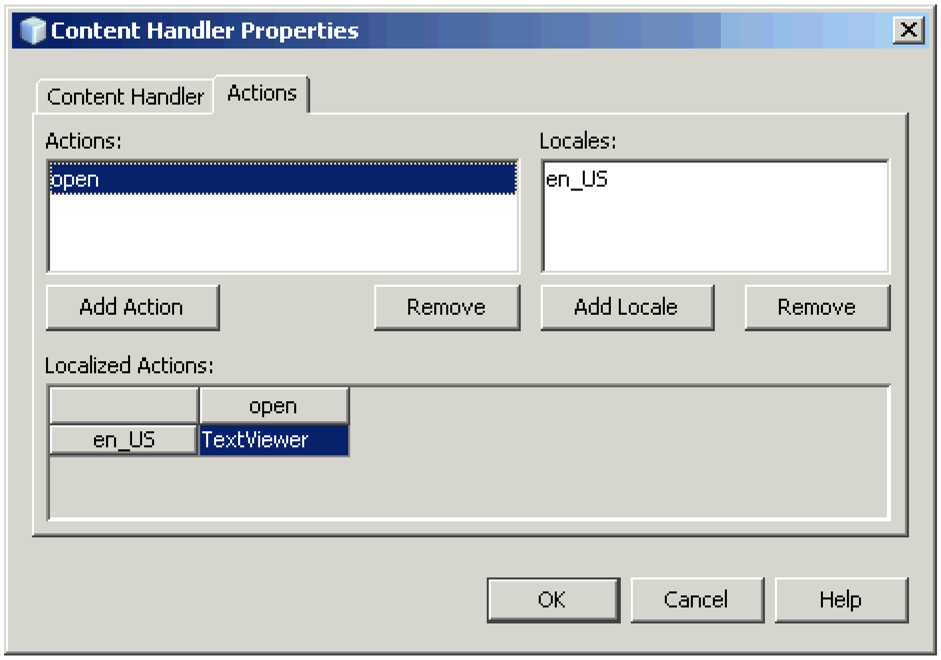
Access allowed to:

- In the Class field, choose a class name from the dropdown menu.
- ID is a required identification string when you invoke a content handler and control access.
- In Content types, list the content types for which this content handler is responsible. Use Add Type and Remove to manage the list.
- In Suffixes, provide a list of URL suffixes that act as a substitute for an explicit content type.
- In Access allowed to, list IDs for content handlers that are allowed access to this content handler. If the list is empty, access to this content handler is granted to every content handler.

25.3 Defining Content Handler Actions

Content handler actions give invoking applications a choice about how to handle content. An Action is associated with an existing content handler. An image viewer content handler, for example, might include an action for viewing the image at its original size and another action that makes the image fill the available screen space.

In the Projects window, right-click on a project and choose Properties from the context menu. In the Category pane, select Application Descriptor, and click the Content Handler tab. Press Add or Edit to open the Content Handler Properties window and click the Actions tab, as shown here.



Actions:
open

Locales:
en_US

Localized Actions:

	open
en_US	TextViewer

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

25.4 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. Go to File > New Project and in the Categories window select Samples > Java ME SDK 3.2 and single-click CHAPIDemo. Click Next.

   Save the sample project in *installdir*`\apps\CHAPIDemo`. Click Finish.

Right-click the project and choose Properties. Choose the Running category, and select Execute through OTA and click OK.

2. Run the project.

   You might see security messages as CHAPIDemo registers itself.

   You might also see a request for permission to use airtime. To speed up the demo interaction, select "Ask once per application use" and select the Yes soft key (if you do not check this option you can still use the demo but you see the airtime message more frequently).

3. Launch CHAPIDemo.

   On the Favorite Links page, choose CHAPI Demo. 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.

4. 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?" Select Yes.

     An authorization Information screen is displayed.
   - Select the Install soft key. The installation is confirmed.

     The installation finishes and you return to the Text Viewer. The Media Handler shows as a separate application in the AMS.

5. 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.

26

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 26.1.3, "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`

26.1 Running SVGDemo

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

Go to File > New Project and in the Categories window select Samples > Java ME SDK 3.2 and single-click SVGDemo. Click Next.

26.1.1 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 8.4, "Oracle Java ME SDK Directories" and Table 8–1. Launch the demo. The application displays the contents of `root1`. Select your SVG file and choose the Open soft key.

26.1.2 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.

26.1.3 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();
```

26.1.4 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.

26.1.5 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.

26.1.6 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.

26.1.7 Picture Decorator

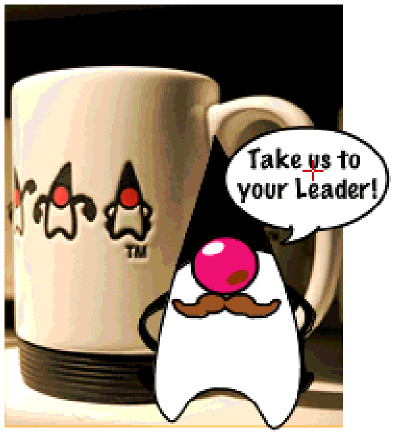
In this sample you use the phone keys to add decorations to a photograph. The key values are:

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

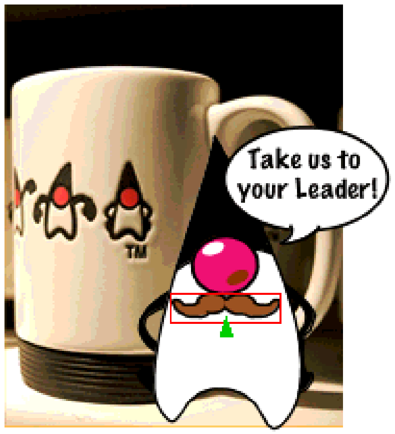
This demo provides 16 pictures for you to decorate.

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.



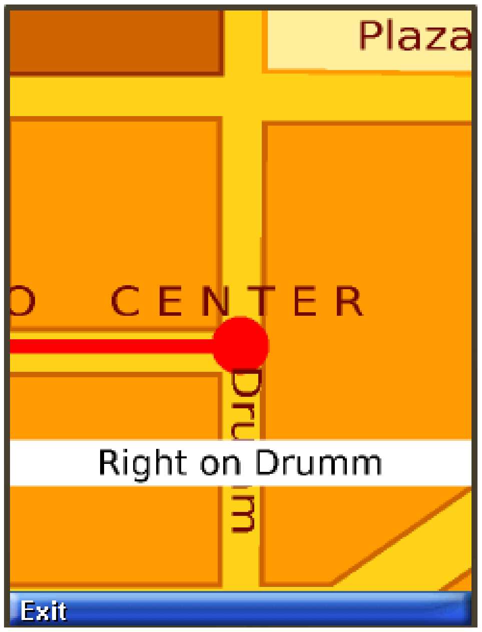
Use the navigation arrows to move the graphic. Use 1 to shrink the graphic, and 3 to enlarge the graphic. Use 5 or 6 to flip, and 7 or 9 to rotate. When you are satisfied with the position, press Select. Look for a green triangle. This is a cursor. Use the navigation keys to move the green triangle around 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.

26.1.8 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.



26.2 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.

27 JSR 239: Java Bindings for Open GL ES

JSR 239 provides a Java language interface to the open standard OpenGL ES graphics API.

27.1 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.

28 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.

28.1 Creating a Mobile Sensor Project

The Mobile Sensor API is automatically included in version 3.2 CLDC projects. In NetBeans, create a new Java ME Mobile Application, choose the CLDC version 3.2 platform, and specify a device that supports CLDC-1.1 and MIDP-2.1 (JavaMEPhone1 for example).

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
```

28.2 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 tab you can view all sensors currently available in the emulator, with the sensor ID, name, and availability. 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 28.3, "Creating a Sensor Script File". See Section 28.4, "SensorBrowser" for a description of how to use the Sensors demo.

28.3 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.

28.4 SensorBrowser

The SensorBrowser application displays the sensor detail information for reach channel defined for the demo.

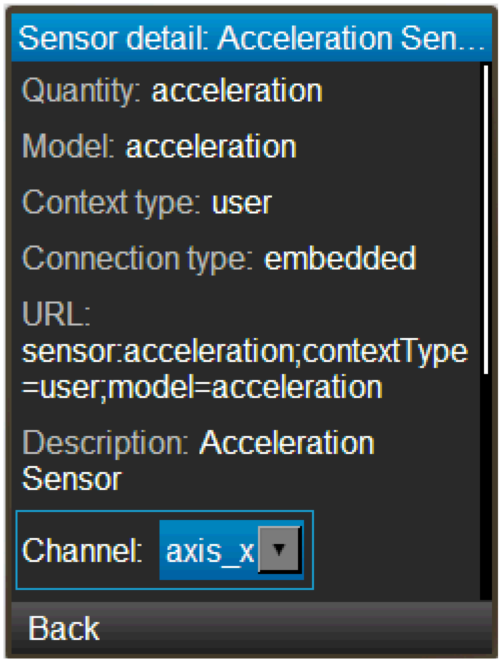
1. In the emulator select SensorBrowser and use the soft key to select Launch the application.

   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.

28.5 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.

29

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*.

29.1 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 29.2, "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 Device > 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 29.2, "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 29.3, "Script Format". In the external events generator, click the Browse button to locate your script, then press Play.

29.2 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>
```

29.3 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 29.2, 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
```

Index