# Java™ Platform Micro Edition Software Development Kit

Version 3.0

Sun Microsystems, Inc.
www.sun.com

April 2009

mesdk-feedback@sun.com

# Contents

CHAPTER 1

# Getting Started

The Java$^{TM}$ Platform Micro Edition Software Development Kit is a natural starting point for learning and using Java ME technology. The focus of this user interface is emulation and deployment. Using this simple yet powerful tool you can create, edit, compile, package, and sign an application. After testing your application in the Java ME Platform 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 Connected Limited Device Configuration (CLDC) and Connected Device Configuration (CDC) platforms and many optional packages (see the list of JCP APIs). In addition, it supplies distribution support for BD-J projects and allows you to run JavaFX™ distributions in the SDK framework.

See the Quick Start and Tips for Legacy Toolkit Users for a brief overview of essential features and concepts.

## Quick Start

The Java ME Platform SDK offers an intuitive user interface. The tips in 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.

  Select Help > Help Contents to open the JavaHelp Online Help viewer. You can also type F1. 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. You can also run the SDK with a different size font, as described in Launching the SDK.

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

  See Running a Project for a general overview of how to run a project.
- See the Projects View and the Files view 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 Working With Projects.
- A project has a default device platform property that is used if you run from the toolbar (the green arrow), the Run menu, or the project's context menu.

  You can run an application on different devices without resetting the main project or changing the default device in the project properties. See Running a Project from the Device Selector.
- The emulator is a remote process, and once it has started it is a separate process from the build process running in the SDK. Stopping the build process or closing a project does not affect the application running in the emulator. You must be sure to terminate the application (the emulator can remain open). See Running a Project.

  The SDK provides two unique instances for most devices. For example, DefaultCldcPhone1 and DefaultCldcPhone2 are the same except for the phone number. This means you can perform tests that require two devices (messaging, for example) without customization. If you want to run more than two emulators you can easily make a copy that preserves the settings you require. See Adding a Device Instance.
- See Creating a CLDC Project and Creating a CDC Project.

See Working With Projects and Tips for Legacy Toolkit Users.

## Tips for Legacy Toolkit Users

If you used the Sun Java Wireless Toolkit for CLDC or the CDC Toolkit in the past, the advice in Quick Start still applies because although the user interface is quite different, the project concept is similar. These 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 matches project requirements and device capabilities for you, so mismatches do not occur.

As mentioned in the Quick Start, clicking the green arrow runs the main project. You can right-click any project and select run.

In the 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 show in the context menu.

- Import applications from legacy toolkits to SDK projects. The installation of the legacy toolkit must exist.

  See Specify WTK Project and Specify CDC Toolkit Project.

- Toolkit **settings** are Application Descriptors in the SDK. Right-click on a project and select Properties. Choose the Application Descriptor category.
- Toolkit **utilities** are generally accessible from the Tools menu in the SDK.

  For example, the WMA console, profiling tools, monitoring tools and more can be started from the SDK Tools menu.

- The emulator is familiar, but there are some fundamental differences.

  It's important to realize that the emulator is now a remote process, and once it has started it is independent of the build process running in the SDK. Stopping the build process or closing a project does not affect the application running in the emulator. You must be sure to terminate the application (the emulator can remain open). For more on this, see Running a Project.

  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 a remote process, the phone number is a property that must be set explicitly for the device instance.

  The SDK provides two unique instances for most devices. For example, DefaultCldcPhone1 and DefaultCldcPhone2 are the same except for the phone number. This means you can perform tests that require two devices (messaging, for example) without customization. If you want to run more than two emulators you can easily make a copy that preserves the settings you require. See Adding a Device Instance.

  The emulator has additional display functionality. See Emulator Options.

See also Quick Start and Working With Projects.

CHAPTER 2

# Java ME Platform SDK Features

The SDK supports three technology platforms, also called stacks. They are: CLDC and MIDP Stack, CDC Stack, and BD-J Stack, as discussed in Emulation Platforms. In addition, it supplies the JavaFX runtime, as discussed in JavaFX Platform.

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 already running, the SDK automatically installs and runs your application. You don't have to launch the emulator over and over.

The SDK supports integration with devices running Windows Mobile and third-party emulators. You can use the SDK to deploy to a real device and perform on-device debugging.

See JCP APIs, Emulation Platforms, and Working With Projects.

## Emulation Platforms

An emulator simulates the execution of an application on one or more target devices. For example, the CLDC platform enables you to run applications on several sample devices with different features, such as screen size, keyboard, runtime profile and other characteristics.

An emulation platform allows you to understand the user experience for an application and test basic portability.

Java ME Platform SDK provides two well-known emulation platforms: CLDC with MIDP and CDC with AGUI. Both platforms include predefined devices with different screen sizes, runtime profiles, and input methods.

The SDK also provides a way to prepare a CDC application for execution on a Blu-ray disc player, as described in BD-J Stack.

The SDK includes the JavaFX runtime and includes two default phone skins for JavaFX. You can use the SDK to run JavaFX project Java application descriptor (JAD) files or Java archive (JAR™) files. See JavaFX Platform.

See CLDC and MIDP Stack, CDC Stack, BD-J Stack, and JavaFX Platform.

## CLDC and MIDP Stack

CLDC/MIDP applications conform to both the Connected Limited Device Configuration and Mobile Information Device Profile (MIDP). The CLDC/MIDP stack is based on the open source phoneME™ Feature project at `https://phoneme.dev.java.net`.

- CLDC 1.1 and MIDP 2.1
- Optimized Mobile Service Architecture (MSA) stack with extensions (JSR 248)
- Java Technology for the Wireless Industry (JTWI) stack (JSR 185)
- All the JSRs listed in TABLE 19-1

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

- A 16-bit or 32-bit processor with a clock speed of 16MHz or higher
- At least 160 KB of non-volatile memory allocated for the CLDC libraries and virtual machine
- At least 192 KB of total memory available for the Java platform
- Low power consumption, often operating on battery power
- Connectivity to some kind of network, often with a wireless, intermittent connection and limited bandwidth

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

See Creating a CLDC Project.

## CDC Stack

A Java ME Platform, Connected Device Configuration 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 `https://phoneme.dev.java.net`. A CDC application conforms to the Connected Device Configuration with a set of profiles that include Personal Basis Profile and AGUI:

- CDC 1.1 with PBP 1.1
- AGUI 1.0

See Creating a CDC Project.

## BD-J Stack

The BD-J stack automates creating a BD-J project that is ready to be burned to a Blu-ray disc or run in an emulator.

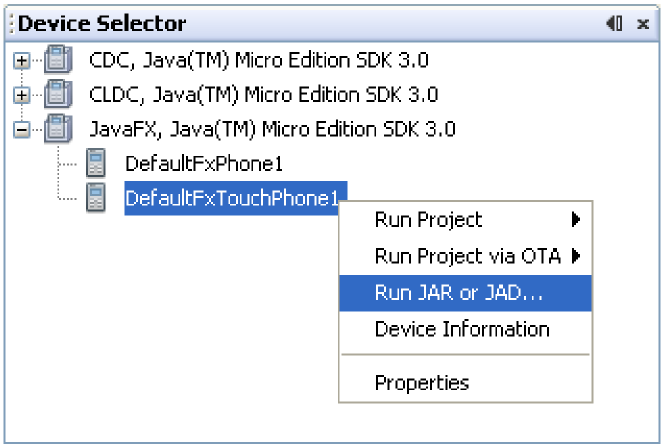
A project wizard simplifies tasks for compiling and bundling the project. The bundled package can be run on an Blu-ray playback software that must be downloaded and installed separately.

See Creating a Stubs for BD-J Platform Project.

## JavaFX Platform

The SDK can run a JavaFX application in JAD or JAR form. The SDK does not support creating or compiling JavaFX projects. The NetBeans™ IDE, combined with the JavaFX SDK, supports JavaFX application development with full editor support, draggable components, and more. Download the development environment from `http://www.javafx.com/`.

When your application is complete, you can use the Java ME SDK to run its JAD or JAR file on an emulator or device. In the Device Selector, right-click on a device and select Run JAR or JAD... from the context menu, then browse to select the file.



See also: `http://www.javafx.com/`

# 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 Platform, and select the Manage Emulators button.

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 register new platforms or add source files and Javadoc™ documents to the existing platforms. For Java ME purposes, the platforms are emulators or SDK platforms for mobile devices.

The Java ME Platform SDK pre-registers CDC, J2ME™ (CLDC and MIDP) and J2SE™ (the JDK serves as the default platform).

The J2SE platform includes the Java SE SDK. The CDC platform includes the CDC and BD-J stacks.

See J2SE and CDC Platforms and J2ME Platforms (CLDC and MIDP).

## J2SE and CDC Platforms

To view the Java Platform Manager, select Tools > Java Platforms. The J2SE platform supports the Java ME Platform SDK. The CDC platform supports the CDC Stack, which also supports the BD-J Stack. In the standard Java ME Platform SDK installation the J2SE and CDC platforms have the same options:

**Classes**. View the platform's classpaths. Add a JAR or folder containing additional classes, moving the classes up and down in the list determines their place in the classpath.

**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 sources files you have added.

See Managing Java Platforms and Adding a Java Platform.

## J2ME Platforms (CLDC and MIDP)

To view the Java Platform Manager, select Tools > Java Platforms. The J2ME platform supports CLDC projects. This platform also serves to run JavaFX projects (see JavaFX Platform):

**Devices**. View all the CLDC devices (including JavaFX devices) that the Device Manager has discovered. Click Refresh to reconfigure the platform and refresh the list.

**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 sources files you have added.

**Tools & Extensions**. View the tools and extensions for this platform.

See Managing Java Platforms, and Adding a Java Platform.

## Adding a Java Platform

To view the Add Java Platform wizard, select Tools > Java Platforms, and click the Add Platform button. You can add platforms that are installed on the SDK's host machine. This wizard sports adding several platforms, however, The wizard is described in the following topics: Selecting a Platform Type, Choose Platform Folder and Name (Java SE and CDC), Choose Java ME MIDP Platform Folders, and Choose Custom Java ME MIDP Platform Emulator.

See Create a Platform for Legacy CDC Projects.

## Selecting a Platform Type

To see this page, select Tools > Java Platforms, and click the Add Platform button. **Select platform type** is the first page of this wizard. The platform types are as follows:

**Java Standard Edition**. The standard Java SDK. There is no need to add SDKs, the Java ME Platform SDK uses only the JDK you specified at installation time.

**Java ME MIDP Platform Emulator**. One or more CLDC emulators that are compliant with the Universal Emulator Interface Specification (UEI) and use technology based on the phoneME™ Feature project (see `https://uei.dev.java.net` and `https://phoneme.dev.java.net`).

**Custom Java ME Platform Emulator**. A platform emulator that is not compliant with the UEI standard.

**Java ME CDC Platform Emulator**. One or more CDC emulators that are compliant with the UEI and use technology based on phoneME™ Advanced (see `https://phoneme.dev.java.net`). The CDC platform includes the CDC and BD-J stacks.

See Selecting a Platform.

## Choose Platform Folder and Name (Java SE and CDC)

To see this page, select Tools > Java Platforms, and click the Add Platform button. Select Java Standard Edition or Java ME CDC Platform Emulator, and click Next.

In the “Look in” field, browse to select a Platform folder. Select a directory and click Next. On the Platform Name page, perform these steps:

1. Specify a platform name.
2. Browse to select the platform sources (typically a zip or JAR file).
3. Browse to choose Javadoc documentation (optional).

Continue to Choose Location (JavaSE and CDC).

## Choose Location (JavaSE and CDC)

This is the final page in the Add Java Platform wizard. If you have chosen a platform, the Platform Name field is populated for you. You might have to wait a second for this to happen. This wizard page also displays the Sources and Javadoc tabs. Click the Add... button to add files to this project. Click Next when you are finished.

See Adding a Java Platform.

## Create a Platform for Legacy CDC Projects

The Java ME Platform SDK version 3.0 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 Java(TM) Platform Micro Edition SDK 3.0". To ensure a successful import, you can create a new platform and give it the legacy name.

The following procedure allows you to import legacy CDC projects without Reference errors (see Resolving Reference Problems).

1. Select Tools > Java Platforms. Select "CDC Java(TM) Platform Micro Edition SDK 3.0", 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_1.0.jar`, `fp_1.1.jar`, `pbp_1.1.jar`, and `secop_1.0.jar`.

   Click Finish, and Close.

See: Importing CDC Projects and Resolving Reference Problems.

## Choose Java ME MIDP Platform Folders

To see this page, select Tools > Java Platforms, and click the Add Platform button. Choose Java ME MIDP Platform Emulator and click Next. In the Platform Folders panel you see the installation directory for the Java ME Platform SDK is detected. If other platforms are installed on your system they can be detected and added to Java ME Platform SDK.

1. To discover additional platforms, click Find More Java ME Platform Folders... The window "Choose directory to search for platforms" opens.
2. This utility finds Java ME platforms and emulators in the scope you specify, and displays them in the Platform Folders window. Choose a folder, or type in a location (for example, `C:\`).
3. If you detected the platform you want, click the box in front of the platform. You see a green check indicating the platform is to be installed. Uncheck any platforms you do not want to install and click Next.

   **Detected Platforms**

   When the platform is discovered and installed, you see three tabs that display information on the platform: Description, Javadocs, and Sources. The Descriptions tab displays platform details, such as supported devices, profiles, configurations, and optional packages.

   Click Finish.

Some emulators might fail to install. If this happens you can try to perform a custom installation as described in Choose Custom Java ME MIDP Platform Emulator.

## Choose Custom Java ME MIDP Platform Emulator

You can use the Add Java Platform wizard to install an emulator that is not UEI compliant. To see this page, select Tools > Java Platforms, and click the Add Platform button. Choose Custom Java ME MIDP Platform Emulator and click Next.

See General Information, Bootstrap Libraries, and Sources & Javadoc.

### General Information

This wizard page defines the general parameters for an emulator that is not UEI compliant. You must supply this information:

**Platform Home**. The path to the directory where the emulator platform is installed. You can enter a path, or use the Browse button to navigate to the directory.

**Platform Name**. A name for the emulator platform.

**Device Name**. A name for the specific device the platform emulates.

**Preverify Command**. The command line syntax that invokes the preverify application to preverify a MIDlet. MIDlet class files must be preverified before they can be run on a device or emulator. The toolkit silently handles preverification during the build process. See the CLDC specification for more information on preverification.

**Execution Command**. The command line syntax that invokes the emulator to execute a MIDlet.

**Debugger Command**. The command line syntax that invokes the emulator to debug a MIDlet.

To see descriptions of the command line syntax parameters, click in the appropriate field. The description is displayed in the area below the debugger command. Click Next when you are finished.

See Running the Emulator From the Command Line.

### Bootstrap Libraries

Based on the General Information you entered, the wizard detects the libraries and APIs the platform or emulator uses. To add libraries, click the Browse button and choose a library file. Click Next when you are finished.

### Sources & Javadoc

Given the files and libraries you have chosen to this point, the wizard detects libraries and APIs. To add additional source files or Javadoc files, click the browse button and choose the files you want to add.

Click Finish to register the emulator platform.

## Support for Third-Party Emulators and Real Devices

Having an emulator does not eliminate the need to test your application on actual target devices. An emulator can only approximate a device's user interface, functionality, and performance. For example, an emulator may not accurately simulate processing speed, so an application may run faster or slower on a target device than it does on an emulator.

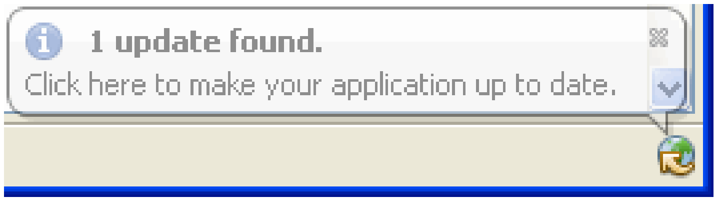
Java ME SDK simplifies deployment to and debugging on real devices running the Sun Java runtime. This version supports Windows Mobile platform based devices, and includes a bundled Java runtime for Windows Mobile devices.

The Microsoft Device Emulator is an example of third-party emulator integration. It means you can deploy applications to Microsoft Device Emulator as easily as you can run on our built-in emulators. See the following topics: CLDC Emulator Installation for a Device Running Windows Mobile and CLDC Installation for Windows Mobile.

## Automatic Update

The Java ME SDK supports automatic updating of individual plugins. The same mechanism can be used to update the entire SDK.

If you have an active Internet connection, the Plugins Manager checks to see whether new plugins or new versions of existing plugins are available. When updates are found you see a notification from the update indicator at the lower right corner of the user interface.


1 update found.
Click here to make your application up to date.

You do not have to wait for a notification. You can select Tools > Plugins and select the Available Plugins tab to see the most current results.

See Using the Plugins Manager and Installing a Plugin Globally.

### Using the Plugins Manager

When you install or update a plugin using the Plugins Manager, the SDK places the plugin `.jar` file and documents in your user directory. See User Directories.

1. Choose Tools > Plugins from the main menu to open the Plugins Manager.
2. Click the Available Plugins tab to display plugins that are available but not installed.
3. In the left pane, select the plugins you want to add and click Install.

4. Complete the pages in the installer to download and install the plugin.

See Available Plugins, Downloaded, Installed Plugins and Plugin Settings.

## Available Plugins

To access this tab, select Tools > Plugins to open the Plugins Manager, then select the Available Plugins tab. To check the update center and refresh the list of available plugins, click Reload Catalog.

## Downloaded

To access this tab, select Tools > Plugins to open the Plugins Manager, then select the Downloaded tab. The left pane displays the manually downloaded plugins that you can install. The right pane displays a description of the selected plugin.

Click Add Plugins to use the file browser to add any plugins that you downloaded to your local system. To install the plugin, select the Install checkbox for the plugin and click Install.

## Installed Plugins

To access this tab, select Tools > Plugins to open the Plugins Manager, then select the Installed tab. The left pane displays a list of the installed plugins. The active column displays the state of the selected plugin.

| | |
|---|---|
| ✓ | The plugin is installed and activated. |
| ✗ | The plugin is installed but deactivated. |
| 🔄 | The SDK needs to be restarted to fully deactivate the plugin. |

When you select a plugin in the left pane, the description of the plugin is displayed in the right pane. You can activate and deactivate installed plugins in the right pane. A deactivated plugin is not loaded on SDK startup. If a plugin is deactivated, it still exists on your local system and you can reactivate the plugin without downloading it again.

An installed plugin can be active or inactive. If a listed plugin is inactive, you might need to install additional plugins to use the plugin.

If you want to completely remove a plugin from your local system, select the checkbox for the plugin in the left pane and then click Uninstall.

## Plugin Settings

To access this tab, select Tools > Plugins to open the Plugins Manager, then select the Settings tab. This tab displays the default update center for the SDK - the Java ME SDK Toolbar Update Center.

By default the Plugins Manager checks for updates once per week, as determined by the Check Interval selection. You can select a different interval from the dropdown list.

Click the Add button to Add an update center URL.

Click the Proxy Settings button to edit the proxy settings to allow access to update centers.

**Force install into shared directories** determines whether the plugin is installed for an individual user or all users in the multiple user environment. If the box is unchecked (default) the plugin is only installed for the user. If it is checked, the installation is global, as described in Installing a Plugin Globally.

## Installing a Plugin Globally

By default the SDK installation is a multiple user environment, as described in User Directories. The typical plugin installation described in Using the Plugins Manager affects only your user directory. To install a plugin globally you must have write access to the SDK installation directory.

1. Choose Tools > Plugins to open the Plugins Manager.
2. Click the Settings tab and then select **Force install into shared directories**.
3. Click the Available Plugins tab, select the Install checkbox for the plugin and click Install. You can also install manually downloaded plugins in the Downloaded tab.
4. Follow the wizard instructions to complete the installation of the plugin.
5. Restart the SDK to activate the new plugin, if necessary.

The SDK places `.jar` files and docs for globally installed plugins in the SDK installation directory instead of in an individual user directory.

See Using the Plugins Manager.

CHAPTER 3

# Using Sample Projects

The Java ME Platform 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.

The source code for every demonstration application is available in the *installdir*/`apps` directory. Subdirectories contain projects, and each project has a `src` directory that contains Java programming language source code.

For example, if the SDK is installed in `C:\Java_ME_Platform_SDK_3.0`, the source code for the SMS sender MIDlet (`example.sms.SMSSend`) in `WMADemo` resides in: *installdir*\apps\`WMADemo\src\example\sms\SMSSend.java`.

For an explanation of the directory structure, see Installation Directories and User Directories. See also: Sample Project Overview, Running a Project, and Running MIDP and CLDC Sample Projects.

## Running a Project

To run a sample project, go to the Start Page tab and single-click a sample project name. The project opens in the Project window and starts running in the emulator.

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

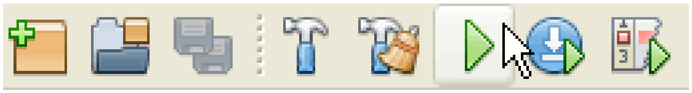
Follow these steps to run a your own projects.

1. Select File > Open Project, and browse to select a project.

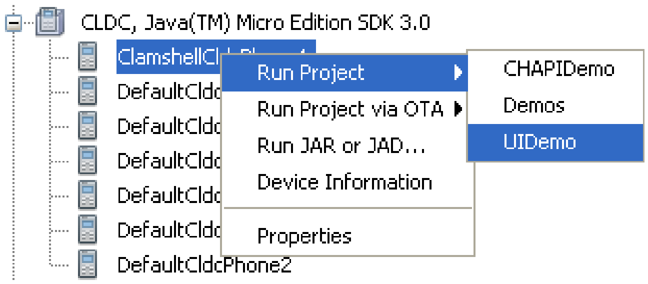
   The project is added to the Projects window.

2. 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. To set a project as main, right-click the project name and select Set as Main Project.

   To run the project on a different device, choose the device in the Device Selector window. Right-click on a device and select Run Project from the context menu. Pull right to see a listing of suitable open projects. If the project you want is not listed, select a different device.

   The device emulator window opens with the demo application running.

3. As the sample project runs, you might need to press one of the soft keys 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.

4. 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.

---

**Note –** Once the emulator is launched, it is an independent process. Pressing the red stop button in the Output window terminates the build script, but it does not close the emulator. Likewise, closing the SDK user interface does not affect the emulator. In the emulator, select Application > Exit to ensure that both the emulator process and the project build process close.

---

## 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 will fail. See 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.

  To perform the installation, MIDlet Suites use *installdir*`/runtimes/cldc-hi-javafx/bin/runMidlet.exe`.

  CDC applications use *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.exe` and `cvm.exe` from checking.

See also Troubleshooting.

## Sample Project Overview

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

Most demonstration applications are simple to run. 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 |
|---|---|---|---|
| AudioDemo | MMAPI 1.1 | Demonstrates audio capabilities, including mixing and playing audio with an animation. | Running AudioDemo |
| BluetoothDemo | JSR 82 | Demonstrates device discovery and data exchange using Bluetooth. | Running the Bluetooth Demo |
| CHAPIDemo | JSR 211 | A content viewer that also uses MediaHandler. | Running the CHAPIDemo Content Browser |
| CityGuide | JSR 179 | A city map that displays landmarks based on the current location. | Running the CityGuide Sample Project |
| Demo3D | JSR 184 | Contains MIDlets that demonstrate how to use 3D graphics, both immediate mode and retained mode. | Running Demo3D Samples |
| Demos | MIDP 2.0 | Includes various examples: animation, color, networking, finance, and others. | Running the Demos Sample Project |
| FPDemo | CLDC 1.1 | Simple floating point calculator. | Running FPDemo |
| Games | MIDP 2.0 | Includes TilePuzzle, WormGame, and PushPuzzle. | Running Games. |
| I18nDemo | JSR 238 | Includes string sorting, number formatting, and a phrase translator. | Running i18nDemo |
| JBricks | JSR 229 | A game that uses the Payment API for buying extra lives or levels. | Running JBricks |
| JSR172Demo | JSR 172 | Demonstrates how to use the JSR 172 API to connect to a web service from a MIDlet. | Running JSR172Demo |
| LWUITDemo | N/A | Demonstrates LWUIT features. | `http://lwuit.dev.java.net/` |
| MMAPIDemos | MMAPI | Demonstrates MMAPI features, including tone sequences, MIDI playback, sampled audio playback, and video. | Running the MMAPI Sample Project |
| Multimedia | MMAPI | Demonstrates different video playback formats. | Running the Multimedia Sample Project |
| NetworkDemo | MIDP 2.0 | Demonstrates how to use datagrams and serial connections. | Running Network Demo |
| PDAPDemo | JSR 75 | Demonstrates how to manipulate contacts, calendar items, and to-do items. Demonstrates accessing local files. | Running PDAPDemo |

**TABLE 3-1** MIDP/CLDC Sample Projects *(Continued)*

| Sample | Optional Package | Description | Instructions |
|---|---|---|---|
| PhotoAlbum | MIDP 2.0 | Demonstrates a variety of image formats. | Running PhotoAlbum |
| SATSADemos | JSR 177 | Demonstrates communication with a smart card and other features of SATSA. | Running SATSADemos |
| SATSAJCRMIDemo | JSR 177 | Shows how to use the SATSA-Java Card Remote Invocation method. | Running SATSAJCRMIDemo |
| Sensors | JSR 256 | The SensorBrowser and Marbles game demonstrate sensor input. | Running the Sensors Sample Project |
| SIPDemo | JSR 180 | Simple message exchange using SIP. | Running SIPDemo |
| SVGContactList | JSR 226 | Uses SVG to create a contact list displayed with different skins. | Running SVGContactList |
| SVGDemo | JSR 226 | Uses different SVG rendering techniques. | Running SVGDemo |
| UIDemo | MIDP 2.0 | Showcases the breadth of MIDP 2.0's user interface capabilities. | Running UIDemo |
| WMADemo | WMA 2.0 | Shows how to send and receive SMS, CBS, and MMS messages. | 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. |
| BdjGunBunny | N/A | A shooting game. | Must be run on an external BD-J platform. See: Compiling, Deploying, and Running a Stubs for BD-J Platform Project. |

## Configuring the Web Browser and Proxy Settings

If you are behind a firewall you might need to 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.
2. Select the General options icon.
3. In the Web Browser field, choose the browser that will be affected by these proxy settings. Click Edit to add or remove a browser from the dropdown list.
4. Choose a Proxy Setting:
   - No Proxy
   - Use System Proxy Settings
   - Manual Proxy Settings

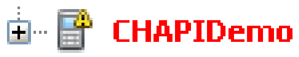
     To set the HTTP Proxy, fill in the proxy server address field and the port number.

     The HTTP Proxy is the host name or numeric IP address of the proxy server to use to connect to HTTP and FTP sites. The Proxy Port is the port number of the proxy server.

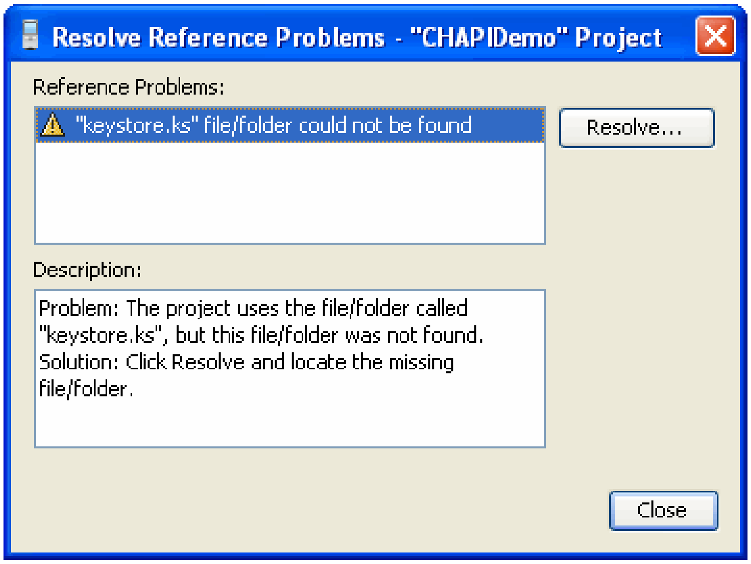
     To set the HTTPS or Socks proxy, click More and fill out the Advanced Proxy Options form.

## 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 file or library that cannot be found. 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 file.

Typically the file `keystore.ks` is stored in *installdir*`\runtimes\cldc-hi-javafx\lib`. Locate and select the file. You receive confirmation that the problem is resolved, and you can now click Close.

## Running MIDP and CLDC Sample Projects

This topic gathers MIDP and CLDC samples that aren't discussed with specific JSRs.

- Running the Demos Sample Project
- Running FPDemo
- Running Games
- Running Network Demo
- Running PhotoAlbum
- Running UIDemo

For other CLDC demos, see TABLE 3-1.

## Running the Demos Sample Project

This demo contains several MIDlets that highlight different MIDP features.

- Colors
- Properties
- Http
- FontTestlet
- Stock
- Tickets
- ManyBalls
- MiniColor
- Chooser
- HttpExample
- HttpView
- PushExample

### Colors

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

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

## 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"
```

## 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 Configuring the Web Browser and Proxy Settings. The HTTP version must be 1.1.
- If you are running antivirus software, you might need to create a rule that allows this MIDlet to allow connections to and from a specific web site. See 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 is able to obtain. If the information fills the screen use the down arrow to scroll to the end. The amount of information depends on the type of request and on the amount of META information the page provides. To provide body information or content, the page must declare CONTENT-LENGTH as described in RFC 2616.

**Using Menu Options**

Use the Menu soft key for the following actions.

- Choose 2 to `GET` information from the selected page.
- Choose 3 to obtain the `POST` information from the selected page.
- Choose 4 to display the `HEAD` attributes for the page.
- Choose 5 to bring up the current list of web pages. You can chose a new page or add your own page to the list. To specify a new URL, choose the Add soft button, then select the Menu soft button and choose OK. The screen displays `http://`. Type in the rest of the URL, making sure to end with a slash (/). For example `http://www.internetnews.com/`. Press the OK soft button. The Http Test screen shows your new URL and prompts for an action.

## 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.

## Stock

Like the Http demonstration, This sample uses an HTTP connection to obtain information. Use the same preparation steps as 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 `JAVA` and select the Done soft key. The stock you added and its current value is now displayed in the ticker. Add a few more stock symbols, such as IBM and HPQ.

### *Change the Update Interval*

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

### *Remove a Stock*

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

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

## Tickets

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

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

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

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

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

---

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

---

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

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

## 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.

## MiniColor

This MIDlet sets an RGB value. It has two modes, a version in which you use navigation keys to change color values, and a virtual version where you press navigation keys to control a virtual keypad.

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.

The virtual keyboard requires an extra step to select each control before you can change its value. In virtual mode, use the navigation keys to highlight a virtual control, then press select to activate the control.

1. Click the Virtual soft key.

   The application displays a 4-way control.

   The up and down keys select a color. The left and right keys lower and raise the value of the selected color.

   Use keyboard navigation keys to choose a control on the display, then press Select.

2. Select the first bar (blue).

   A white box surrounds the selected color.

   Blue and has a value of 0 so you don't see any blue yet.

3. Choose the right control and press select.

   Each click raises the value by 32.

## 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.

### HttpExample

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

### HttpView

This application displays a predefined URL. You can also enter a new URL.

- Launch the HttpView application.
- Select Menu > 2 to get the contents of the URL.
- Select Menu > 6 to view application instructions.

### PushExample

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

## 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. Press the Calc soft button to calculate the result.

## 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. 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. Once 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.

## 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 DefaultCldcMsaPhone1 and DefaultCldcMsaPhone2.

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

   For example, in the client, type `Hello Server` In the Send box. Choose the Send soft key. The emulator activates a blue light during the transmission.
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`. Select Send.
4. Back in the Socket Client, the status shows the message received from the server. Until you send a new message, the Send box contains the previous message you sent.

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

## Running PhotoAlbum

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

## 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**. Any symbol

With the exception of Predict, these categories act as filters. For example, if you assign 123 to a field and you type "abc", nothing is entered in the field.

**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 five input types

**DateField**. Set date and time using drop lists.

**Ticker**. A scrolling ticker.

CHAPTER 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.

The Java ME Platform SDK creates its project infrastructure directly on top of Apache Ant. With this infrastructure in place, you can build and run your projects within the SDK or from the command line. The build process is controlled by project properties, as described in Building a Project.

## About Projects

A project contains all the supporting files for one application. When a project is created, the SDK performs these tasks:

- Creates a source tree you can examine in the Working With Projects or Viewing Project Files views.
- 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. The build process is controlled by project properties, as described in Building a Project.

The SDK provides two views of the project:

- The Working With Projects window provides a logical view of the project.
- The Viewing Project 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.

View these topics to learn about project properties: Creating a CLDC Project, Creating a CDC Project, Working With Projects, Viewing Project Files and Adding Files to a Project.

See also: Building a Project, Building a Project from the Command Line.

## CLDC Projects

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

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 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.

When the application is run, the name of the main application class is passed to the Java virtual machine. This class must include a method named `main()` that handles the application's class loading, object creation, and method execution. The project manages its own life cycle and system resource needs. When the `main()` method exits, the application terminates.

See Creating a CLDC Project, Working With Projects, and Viewing Project Files.

## 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.

Java ME Platform SDK version 3.0 supports CDC projects running as standalone applications. This means 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.

---

Like MIDP projects, 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.

See Creating a CDC Project, Creating a Stubs for BD-J Platform Project, Working With Projects, and Viewing Project Files.

## Creating a CLDC Project

A CLDC project uses the MIDP application template and preselects devices that support MIDP and CLDC. The SDK provides a wizard for creating new projects quickly and easily. Most project properties can be edited later by changing the project properties.

The project provides a basic infrastructure for CLDC development. You provide source files, resource files, and project settings as needed.

1. Select File > New Project.

   The New Project wizard opens. Java ME SDK is the only category.

2. Follow the prompts in the New Project wizard, consulting Help if necessary. See Choose Project, Name and Location, Platform Selection, and Specify WTK Project.

3. To run the new project, follow the steps in Running a Project, except select your new project instead of a sample project.

4. Be sure to exit or close the application when you are finished.

   Once the emulator is launched, it runs as an independent process. Pressing the red stop button in the SDK user interface or closing the SDK does not stop the application running in the emulator.

   Applications usually provide a way to terminate. For example, most of the samples offer an Exit soft key, or an option in the soft menu. You can close the application and leave the emulator running (so you do not have to wait for the emulator to open the next time you run the project).

   If you want to close the emulator and stop the project build process, select Application > Exit.

Now that the project has been created, you can modify its properties as described in the following topics: Viewing General Project Properties, Selecting a Platform, Editing Application Descriptor Properties, Building a Project, Running a MIDP Project.

## Choose Project

This is the first page in the New Project wizard. For MIDP the project options are as follows:

**MIDP Application**. Create a new MIDP application in a CLDC/MIDP project.

**Import Wireless Toolkit Project**. Create a project by importing a Sun Java Wireless Toolkit project from an existing toolkit installation.

For CDC the project options are as follows:

**CDC Application**. Create a new CDC application in a CLDC project.

**Import CDC Toolkit Project**. Create a project by importing a CDC Toolkit project from an existing toolkit installation.

## Name and Location

This is the second page in the New Project wizard.

**Project Name**. Enter a project name. If you are importing an existing project this field is prepopulated with the old filename prefixed.

**Project Location**. The default location is `C:\Documents and Settings\`*user*`\My Documents\JavaMESDKProjects`.

**Project Folder**.The Project Folder value is extrapolated from Project Name and Project Location.

**Set as Main Project**. Check this box to make the project the Main Project when it is first opened. The Main project is automatically the focus of all actions initiated from the user interface (for example, the actions on the Run menu, which provide the same functionality as clicking icons on the main tool bar).

**Create Hello MIDlet**. This check box is only visible for a new MIDP project. It inserts sample MIDlet code as a template for your development. You can compile and run the MIDlet immediately.

**Create Main Class**. This check box is only visible for a new CDC project. Enter the fully qualified name of the main class without the `.java` extension. For example: `com.me.MyClass`.

## Platform Selection

You can view this form in the New Project wizard, or, in the Projects view. Right-click a project, select Properties, and select Platform.

These settings help you test how your project runs on devices with different capabilities. Your choice of device limits your choice of Device Configuration, Device Profile, and Optional Packages (if applicable).

**Emulator Platform**. In the New Project wizard this field is predetermined.

**Device**. Select a device. Only devices appropriate for the platform appear in the Device drop-down menu. The device selection determines the remaining options.

**Device Configuration**. Select a CLDC version.

**Device Profile**. Select a MIDP version. The available selections are determined by the Device Configuration.

**Optional Packages**. This pane is visible when you are viewing an existing project. You can check or uncheck optional packages to approximate device capabilities.

### Specify WTK Project

To see this form, start the New Project wizard and select Import Wireless Toolkit Project.

**WTK Location**. Browse to select the location of your Sun Java Wireless Toolkit installation. Choose the installation directory.

**Detected Applications**. When the WTK Location is selected the Detected Applications window displays the available projects. Highlight a project, and click Next.

See Importing MIDP Projects.

## Creating a CDC Project

The SDK provides a wizard for creating new projects quickly and easily. Most project properties can be edited later on. CDC core, FP, and PBP APIs are automatically included in every CDC project.

1. Select File > New Project.

   The New Project wizard opens.

2. Follow the prompts in the New Project wizard, consulting help if necessary. See Choose Project (CDC), Name and Location, Platform Selection (CDC), and Specify CDC Toolkit Project.

3. The Name and Location page has the following fields:

   **Project Name**. The name you supply is the default name for the Main class, if you use one.

   **Project Location**. Browse to the project location. The default is `/JavaMESDKProjects`.

**Project Folder**. This value is extrapolated from the Name and Location entries.

4. **Set as Main Project**. Check this box to set this project as main. Toolbar actions, such as Build and Run, operate on the main project. The main project is displayed in bold font in the project tree.
5. **Create Main Class.** If you want to create a sample Main class in the project, check the box and supply a project name. If the box is not checked, the project will not have a Main class.
6. Select platform.

   Select the platform, a device, and the profile. Click Finish.

   If an AGUI device is selected, the AGUI API is added to the project.
7. To run the new project follow the steps in Running a Project, except you can select your new project instead of a sample project.

   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.

---

**Note –** Once the emulator is launched, it runs as an independent process. Pressing the red stop button in the Output window terminates the build script, but it does not close the emulator. Likewise, closing the SDK does not affect the emulator. In the emulator, select Application > Exit to ensure that both the emulator and the project build process close.

---

To modify the project, right-click on the project node and select Properties.

See the following topics: Viewing General Project Properties, Selecting a Platform, Editing Application Descriptor Properties, Building a Project, and Running a CDC Project

## Choose Project (CDC)

You can view this page in the New Project wizard. The CDC project options are as follows:

**CDC Application**. Create a new CDC application in a CLDC project.

**Import CDC Toolkit Project**. Create a project by importing a CDC Toolkit project from an existing toolkit installation.

## Platform Selection (CDC)

You can view this form in the New Project wizard, or, in the Projects view, right-click a project, select Properties, and select Platform.

These settings help you test how your project runs on devices with different capabilities. Your choice of device limits your choice of Device Configuration and Device Profile.

**Java Platform**. A choice of CDC or Stubs for BD-J Platform. Stubs for BD-J Platform is a special type of CDC platform.

**Device**. Select a device. Only devices appropriate for the platform appear in the Device drop-down menu. The device selection determines the remaining options.

**Profile**. PBP-1.1 is the only option.

See Creating a CDC Project and Creating a Stubs for BD-J Platform Project.

## Specify CDC Toolkit Project

To see this form, start the New Project wizard and select Import CDC Toolkit Project.

**Project Location**. Browse to select the location of your legacy CDC Toolkit project.

See Importing CDC Projects.

# 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. Selecting New > Other allows you to add different types of files to the project. For a sample procedure, see Generating Stub Files from WSDL Descriptors.

**Build**. Builds a distribution Java archive (JAR) file. The build process is controlled by project properties, as described in Building a Project.

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

**Clean**. Cleans the build files.

**Run**. Runs the project with the default device, as specified on the Selecting a Platform property page.

**Set as Main Project**. Toolbar actions, such as clicking the green Run button, act upon the main project.

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

The **Source Packages** node encapsulates all the Java packages in the project. Right-click on the Source Packages node and choose New to add a new MIDlet to your application.

The **Resources** node encapsulates all resources and libraries of the active configuration. Right-click the Resources node to add Projects, JARs, folders, and libraries as resources for your application. You cannot add or remove inherited resources.

See also: Viewing Project Files, Creating a CLDC Project, Creating a CDC Project, Creating a Stubs for BD-J Platform Project, About Projects, and Importing CDC Projects.

## Viewing 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 will be 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 will 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 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 IDE-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, it is not recommended. Do not edit genfiles.properties.

**res**.Resource files you have added to the project. See 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: Creating a CLDC Project, Creating a CDC Project, Creating a Stubs for BD-J Platform Project, and Importing CDC Projects.

## Creating 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 of the new MIDP class.

**Midlet Class Name.** The name that users see when the application runs on a device.

**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. 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 location of the MIDlet.

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

## Importing MIDP Projects

If you created a project using the Sun Java Wireless Toolkit for CLDC 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 toolkit installation home. Use browse to open the top-level installation directory.
4. The wizard detects any applications in the legacy installation and displays their locations on disk. Select a project and click Next.
5. Supply the Project Name, Location, and Folder for the new project (see `/JavaMESDKProjects`). Note that 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 Managing Keystores and Key Pairs.

## Importing CDC Projects

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.

---

1. The CDC platform name for the Java ME Platform SDK version 3.0 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. To avoid this error, create a platform with the legacy name, as described in Create a Platform for Legacy CDC Projects.

   You only need to do this once.

2. Select File > New Project.

3. In the Projects area select the import action for CDC Toolkit. Click Next.

4. 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.

5. 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: Create a Platform for Legacy CDC Projects, Viewing Project Files

## Adding 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 Adding Libraries and Resources.

## Find in 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://java.sun.com/javase/6/docs/api/java/util/regex/Pattern.html#sum

CHAPTER 5

# Viewing and Editing Project Properties

To view project properties, right-click the project node and select Properties. This tool allows you to customize the following project properties:

- Viewing General Project Properties
- Selecting a Platform
- Editing Application Descriptor Properties
  - MIDP Attributes
  - MIDlets
  - Push Registry
  - API Permissions
  - See also Using Content Handlers
- Building a Project
  - Compiling
  - Adding Libraries and Resources
  - Creating JAR and JAD Files (Packaging)
  - Obfuscating
  - Signing
- Running Settings

## Viewing General Project Properties

To view this property page, right-click on a project and 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 Adding Libraries and Resources.

## Selecting a Platform

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.

CDC and CLDC/MIDP platform types are listed in the Select Platform Type dropdown. The emulator platform is supplied based on the platform type. The Stubs for BD-J Platform is a type of CDC platform. See Creating a Stubs for BD-J Platform Project.

Select a platform type from the dropdown menu.

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.

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

See: Creating a CLDC Project, Creating a CDC Project, and Creating a Stubs for BD-J Platform Project

## 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, as discussed in the following topics:

CDC Attributes and MIDP Attributes, MIDlets, Push Registry, API Permissions, and Using Content Handlers

### CDC Attributes

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

**Application Name**. Enter 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 descriptionof the application.

### 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.

**Name**. The name of the attribute.

**Value**. The values for each attribute.

To avoid errors in verification:

- Make sure all required attributes have a defined value.
- Do not begin user-defined attribute keys with `MIDlet-` or `MicroEdition-`.

See: Editing an Attribute, and Adding an Attribute

## Adding 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. Do not begin a user-defined attribute name with `MIDlet-` or `MicroEdition-`.
3. Enter a value for the attribute.
4. Click OK.

## Editing an Attribute

Follow these steps to 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. See: MIDlets, Push Registry, and Signing.

## Removing an Attribute

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

## 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.

## Adding a MIDlet

Follow these steps to add a MIDlet.

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

## Editing a MIDlet

Follow these steps to 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.

## Removing a MIDLet

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

## Changing 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.

## 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.

### Adding a Push Registry Entry

Follow these steps to edit a Push Registry entry.

1. Click Edit... to open the Edit 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. To view the JAD file, choose the Application Descriptor Property settings from the left pane.

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

### Remove a Push Registry Entry

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

### 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.

## 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.

The Requested Permissions table in this page has two fields:

- **API**. The API permissions requested by the MIDlet.
- **Required**. If checked, the permission is required. At installation, if the required permission can not be granted by the application management software, the application will not be installed. If the box is not checked, the permission is optional. If an optional permission is denied, the application might continue to function, although its functionality can be limited.

**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. In the Requested Permissions table, check the Required box if you want installation to fail in the event 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/.

## 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:

- Compiling
- Adding Libraries and Resources
- Obfuscating
- Creating JAR and JAD Files (Packaging)

- Signing

## 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. For example, for the standard Ant 1.7.1 release, the Ant installation directory is `ant/lib/apache-ant-1.7.1`. If you enter a directory that does not match this structure, the SDK gives you an error.

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

**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}**. If you are compiling using Jikes (`build.compiler=jikes`), setting this property to true enables Emacs-compatible error messages. It is recommended that you leave this property set to true even if you are not using Jikes, since the SDK prefers Emacs-compatible error messages.

## 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`).

**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.

## 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.

This page allows you to add a dependent project, libraries, and other supporting files to the current project.

**Add Project**. The JAR file produced by another project, as well as 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.

Once 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.

## 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. Does not apply to CDC projects.

**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.

## 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. The default obfuscator included with the SDK is ProGuard. You can find more details about command parameters for this obfuscator at: http://proguard.sourceforge.net.

## 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 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 for use.

**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 Managing Keystores and Key Pairs and Exporting a Key.

---

**Note –** CDC projects cannot be signed with this tool. 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
```

---

## Exporting a Key

To view this dialog, right-click on a project and choose Properties. In the Properties window Build category, choose Signing and click Export key into Java ME SDK/Platform/Emulator. Or, Select Tools > Keystores, then select a keystore and a key, and click Export.

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 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.

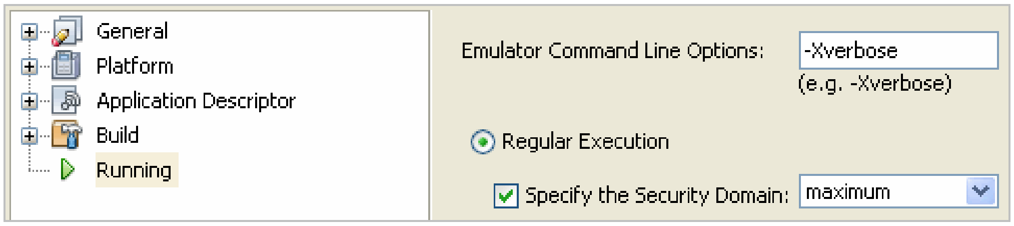
## Running Settings

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 Running a MIDP Project and Running a CDC Project.

### Running a MIDP Project

To set emulator command line options for a MIDP project, type in the command line switches. See Running the Emulator From the Command Line.

For CLDC projects, the Regular execution button is enabled by default. This means 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.



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

● Regular Execution

✓ Specify the Security Domain: maximum

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 Security Domains and Specifying the Security Domain for a Project.

### Running a CDC Project

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 this means 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 Running Settings and CDC Options.

CHAPTER 6

# Running Projects in the Emulator

The Java ME Platform SDK emulator simulates a MIDP device on your desktop computer. The emulator does not represent a specific device, but it provides correct implementations of its supported APIs. The SDK uses the device manager to detect devices and displays the available devices in the Device Selector window. See Running the Device Manager.

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

If the Device Selector window is not visible, select Window > Device Selector.

See also:

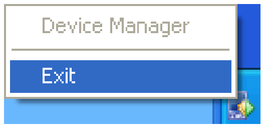
- Viewing Device Properties
- Setting Device Properties
- Running a Project from the Device Selector

## Understanding the Emulator

When the Java ME Platform SDK is launched the Device Manager is automatically launched to detect devices. 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`.

The Device Manager icon looks like this:

You can right-click on the icon to exit the service.

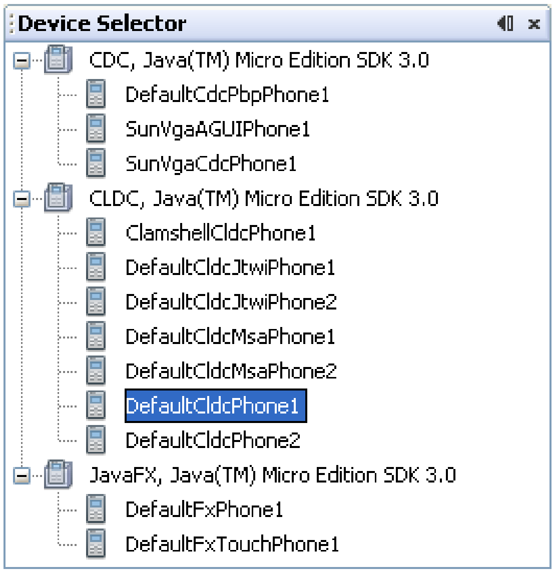


See also: Platform Properties, Device Information, Device Properties, Running the Emulator From the Command Line, and Running the Device Manager

## Viewing Device Properties

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

If no Java ME platform is registered in the toolbar, 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 just need to wait a few seconds.



Each sub node represents an emulator skin for a device. Two instances are provided for some CLDC devices, for example, DefaultCldcPhone1 and DefaultCldcPhone2. These devices have the same capabilities but unique phone numbers, making it easy for you to test communication between two devices.

The properties for each device skin are stored in XML files in your user work directory. See "`/javame-sdk/3.0/work`" on page 80, and TABLE 8-1.

See also: Platform Properties, Device Information, and Device Properties

## Platform Properties

To view platform properties from the device selector, right-click on the platform node (for example, CDC) and select Properties. The platform properties display in a separate window.

If you have selected Window > Properties, the Properties window is, by default, docked in the upper right portion of the user interface. Selecting a node in the Project or Files trees causes any available properties to be displayed.

## Device Information

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

## Device Properties

In the Device Selector window, right-click on the platform node and select Properties. The device properties display in a separate window.

If you have selected Window > Properties, the Properties window is docked in the SDK. Selecting any node causes its properties to be displayed.

See Setting Device Properties.

## 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.

**Debug Port**. The debugging port number. A default is supplied but it can be changed.

**Enable Profiler**. Check this box to enable profiling. This is a CPU snapshot collected during emulation. See Saving Profiler Data.

If you want a profile of the Java Heap contents during the virtual machine execution, see Virtual Machine Memory Profiler (Java Heap Memory Observe Tool).

**Enable Network Monitor**. Check this box to enable the network monitor.

**Phone Number**. 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. If the emulator is running and another emulator is opened, the phone number for the second instance is incremented by one.

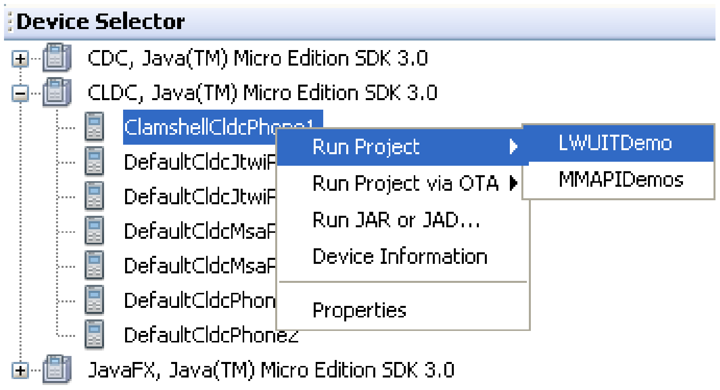
**Heapsize**. The heap is the memory allocated on a device to store your applications's objects. The Heapsize property is the maximum heap size for the emulator. You can choose a new maximum size from the dropdown menu.

**Security Domain**. Select a security setting from the dropdown menu. See Security Domains. Applies to CLDC and JavaFX platforms.

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

## 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. If projects are not suitable they are displayed in gray font.



You can also launch the emulator to run a project from the command line, as explained in Running the Emulator From the Command Line.

## Running Projects Simultaneously on a Single Device

MSA-compliant 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 an MSA-compliant device and run Games. When the emulator launches run AudioDemo on the same device.

   As each MIDlet loads, the AMS automatically launches 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 SunSamples - 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 from the soft menu, select 1, Launch. 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 select Switch to. You can press the Play soft button to resume the Audio Player.

## Emulator Options

The emulator has Application, View, and Help menus.

The Application menu is only populated for CLDC and JavaFX platforms. The Application options are as follows:

| Option | Accelerator | Description |
|---|---|---|
| AMS Home | F4 | Exit the current application and return to the Application Management Software home. |
| Change Locale | | This option only works with localized MIDlets.<br>Enter a locale identifier. The format is similar to that of the Supported Locales for Java SE 6, as follows:<br>2-letter-lang-code *separator* 2-letter-country-code<br>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. |
| 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 view. |
| Exit | | Close the emulator process and stop the build process (or processes). |

The View menu is available in full to CLDC and JavaFX platforms. For 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. |
| External Events Generator | The external events generator is a utility for simulating external file systems or applications communicating with the application running on the emulator. Inputs can come from a script file on your PC or they can be injected based on your interaction with a user interface.<br>The external events generator provides a tab to support JSR implementations that require external event emulation: File Connection (75), Location (179), Payment (229), and Sensors (256). See FileConnection API, Running the CityGuide Sample Project, Running JBricks, Using a Mobile Sensor Project. |
| Orientation | Choose a degree of rotation (0, 90, 180, or 270 degrees clockwise from the 0 position) or, rotate clockwise by 90 degrees from the last position (F8) or counterclockwise by 90 degrees (F9). |

## Adding a Device Instance

As described in Viewing Device Properties, a particular device emulator can have more than one instance, and each instance is differentiated by a number appended to the emulator name, as seen in TABLE 8-1. Each device instance is stored in a numbered directory in `/javame-sdk/3.0/work`.

To create your own instance, follow these steps:

1. Close the Java ME Platform SDK.
2. In `/javame-sdk/3.0/work`, copy a numbered directory and rename it with the next number in the sequence.
3. In the new directory, open the `properties.xml` file and change the `name` property string to a unique name.

   You can also change the values in `device.properties`.
4. In the system tray, right-click on the Device Manager icon and select Exit from the context menu.

5. Start the Java ME Platform SDK.

In the Device Selector you see a new node named Other. All your custom devices are listed here. To assign this device to a project, right-click the project, select Properties, and choose Platform. Your instance appears in the Device drop list.

You can also edit the device adaptor to create a new instance. For example, to create a second instance of the ClamshellCldcPhone, follow these steps:

1. Go to *installdir*`\toolkit-lib\process\device-manager\device-adapter\ClamshellCldcPhone`.
2. Make a copy of `1.bean`, and name it `2.bean`.
3. Edit `2.bean` to change the device number to 2. For example, ClamshellCldcPhone2.
4. Exit the SDK and exit the Device Manager.
5. Start the SDK. ClamshellCldcPhone2 is listed in the Other category.

CHAPTER 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.

See: Searching for Devices, Filtering the WURFL Search

## Searching for Devices

Follow these steps to search the WURFL database:

1. Select Tools > 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 Filtering the WURFL Search.

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.

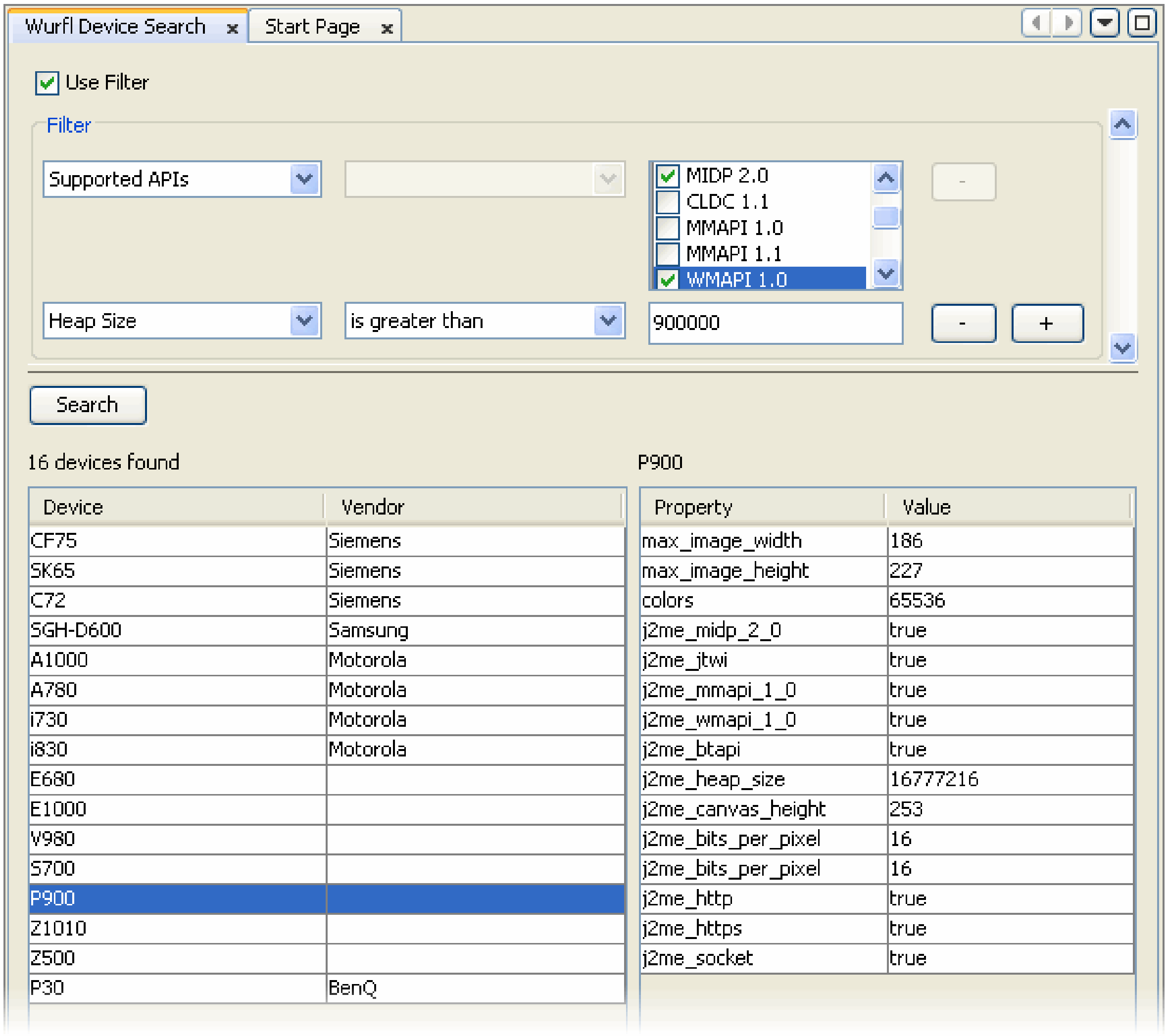
4. To add another search criteria, click the + button.

   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 Filtering the WURFL Search.

## Filtering the WURFL Search

As discussed in Searching for Devices, you can use the filter to set search constraints. You should set at least one constraint.

**Supported Properties**

This utility searches on a predefined list of constraints that have corresponding properties in the Java ME Platform 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
  - Localization 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 contstraints, go to:
`http://wurfl.sourceforge.net/help_doc.php`.

See also Searching for Devices.

CHAPTER 8

# Finding Files in the Multiple User Environment

The Java ME Platform SDK can be installed on a system running a supported version of Windows. 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 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 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. See User Directories.

- Installation Directories
- User Directories

To locate logs, see Java ME Platform SDK GUI Logs, Device Manager Logs, and Device Instance Logs.

## 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 Understanding the Emulator. A different user can then launch the SDK and own all processes.

## Installation Directories

The Java ME SDK installation directory structure conforms to the Universal Emulator Interface Specification, 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:

- `apps`. Contains examples for supported platforms:
  - BD-J: BdjGunBunny
  - CDC and AGUI: AGUIJava2DDemo and AGUISwingSet2 as described in TABLE 3-2.
  - CLDC and MIDP: all others, as shown in TABLE 3-1.
- `bin`. Contains the following command line tools.
  - `cref`.Java Card simulator for working with SATSA JSR 177. See Java Card Platform Simulator (`cref`).
  - `device-manager`. The device manager is a component that must be running when you work with Java ME Platform SDK. After installation it starts as a service, and it will automatically restart every time your computer restarts. When it is running, it appears in the Windows system tray as an icon. Right-click on the icon to view its controls.
  - `device-address`. `device-address.exe` is a tool for viewing, adding, and removing devices that the SDK is not able to discover automatically. See Managing Device Addresses (device-address).
  - `emulator`. UEI compliant emulator. See Running the Emulator From the Command Line.
  - `jadtool`. Tool for signing MIDlets. See Signing MIDlet Suites (`jadtool.exe`).
  - `mekeytool`. Management of ME keystores. See Signing MIDlet Suites (`jadtool.exe`).
  - `payment-console`. Minimalistic console for viewing payment transactions. An equivalent tool exists in the Java ME SDK user interface.
  - `preverify`. The Java ME preverifier.
  - `resourcesmanager`. A tool for managing JSR 238 resource bundles. An equivalent tool exists in the Java ME Platform SDK user interface.
  - `runBDJ`. Run a BD-J project in a player.
  - `wma-tool`. A command line tool for sending and receiving SMS, CBS, and MMS messages. See Running WMA Tool.
  - `wscompile`. Compiles of stubs and skeletons for JSR 172. See Generating Stubs (wscompile).

- `docs`. Release documentation.
- `lib.` JSR JAR files for compilation.
- `on-device`. Windows Mobile Java Runtime for ARM.
- `toolbar`. A simple development environment.

## User Directories

At installation time you are prompted to specify a location for user files. The default location is:

`C:\Documents and Settings\`*user*`\javame-sdk`

If you do not remember the location you specified, select Help > About in the main window. This documentation sometimes uses *users.home* to represent this path.

You cannot choose another location after the installation. If you desire a different directory, you must reinstall.

The user directories are: `/javame-sdk`, `/javame-sdk/3.0/work`, and `/javame-sdk/toolbar/3.0`.

### /javame-sdk

This directory contains device instances and session information. If you delete this directory, it will be recreated automatically when the device manager is restarted.

## /javame-sdk/3.0/work

The work directory contains device instances in directories 0-11. Each instance has, at a minimum, its own log file and a `device.properties` file. The directory number is also the ID number assigned to the emulator skin. It is displayed in the emulator's title bar.

**TABLE 8-1** Device Names

| Directory | Device |
|---|---|
| 0 | ClamshellCldcPhone1 |
| 1 | DefaultCdcPbpPhone1 |
| 2 | DefaultCldcJtwiPhone1 |
| 3 | DefaultCldcJtwiPhone2 |
| 4 | DefaultCldcMsaPhone1 |
| 5 | DefaultCldcMsaPhone2 |
| 6 | DefaultCldcPhone1 |
| 7 | DefaultCldcPhone2 |
| 8 | DefaultFxPhone1 |
| 9 | DefaultCdcPbpPhone |
| 10 | SunVgaAGUIPhone1 |
| 11 | SunVgaCdcPhone1 |

Profiling creates the following data file in a device instance directory:

`/javame-sdk/3.0/work/`*device*/data.prof.

## /javame-sdk/toolbar/3.0

Contains property files and configuration files for the GUI development environment.

## /JavaMESDKProjects

The default project directory is `C:\Documents and Settings\`*User*`\My Documents\JavaMESDKProjects`.

CHAPTER 9

# Profiling 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 a profiler data snapshot is written to a `.prof` you can load and view in the SDK. As you view the snapshot you can investigate particular methods or classes and save a customized snapshot (a `.nps` file).

---

**Note –** This feature might slow the execution of your application.

---

Continue to:

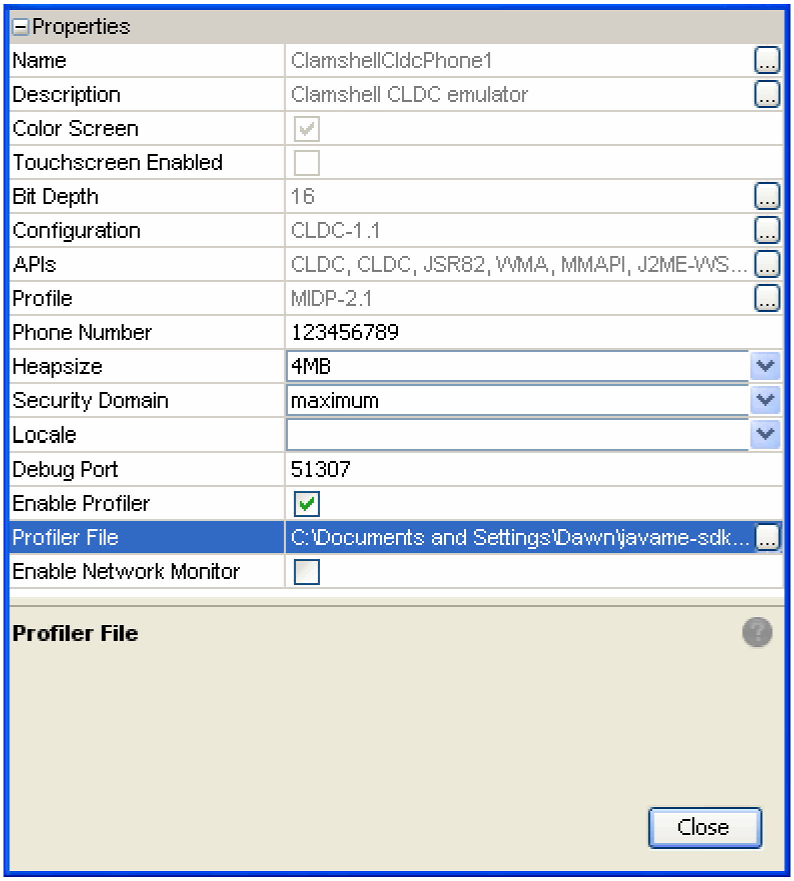
- Saving Profiler Data
- Loading Profiler Data
- Viewing Profiler Data
- Saving Customized Snapshots and Images
- Loading a Customized Snapshot

## Saving Profiler Data

Follow these steps to enable data collection:

1. In the Device Selector window, right-click on a device and choose Properties.
2. Check the Enable Profiler option, and note the location of the profiler output file.

You can also edit the `device.properties` file to set `profiler.enabled` to `true`. The properties file is located in the device instance directory, as described in `/javame-sdk/3.0/work`. The device number corresponds to a device in the device selector, as discussed in TABLE 8-1.



Properties

| | |
|---|---|
| Name | ClamshellCldcPhone1 |
| Description | Clamshell CLDC emulator |
| Color Screen | ✓ |
| Touchscreen Enabled | |
| Bit Depth | 16 |
| Configuration | CLDC-1.1 |
| APIs | CLDC, CLDC, JSR82, WMA, MMAPI, J2ME-WS... |
| Profile | MIDP-2.1 |
| Phone Number | 123456789 |
| Heapsize | 4MB |
| Security Domain | maximum |
| Locale | |
| Debug Port | 51307 |
| Enable Profiler | ✓ |
| Profiler File | C:\Documents and Settings\Dawn\javame-sdk... |
| Enable Network Monitor | |

Profiler File

---

**Note –** It's helpful to display the output window. If it's not open, select Window > Output > Output.

---

3. Start your application.

   Interact with your application as you normally would.

4. In the emulator, select Application > Exit.

   The profile data snapshot is saved, and the SDK reports the location of the profile data in the Output window. The profile data is displayed in a tab labeled CPU:*time*, where *time* is the time the snapshot was saved.

See Loading Profiler Data.

## Loading Profiler Data

This procedure views data collected with the Profiler, as described in Saving Profiler Data.

Follow these steps to retrieve profile data:

1. In the main window toolbar, select Tools >Import Java ME SDK Snapshot...
2. Choose the `data.prof` file.

   The default location is: `C:\Documents and Settings\`*user*`\javame-sdk\3.0\work\`*device-number*`\data.prof`.

   The Profiler opens in its own tab in the main window labeled CPU:*time*.

See Saving Profiler Data and Viewing Profiler Data.

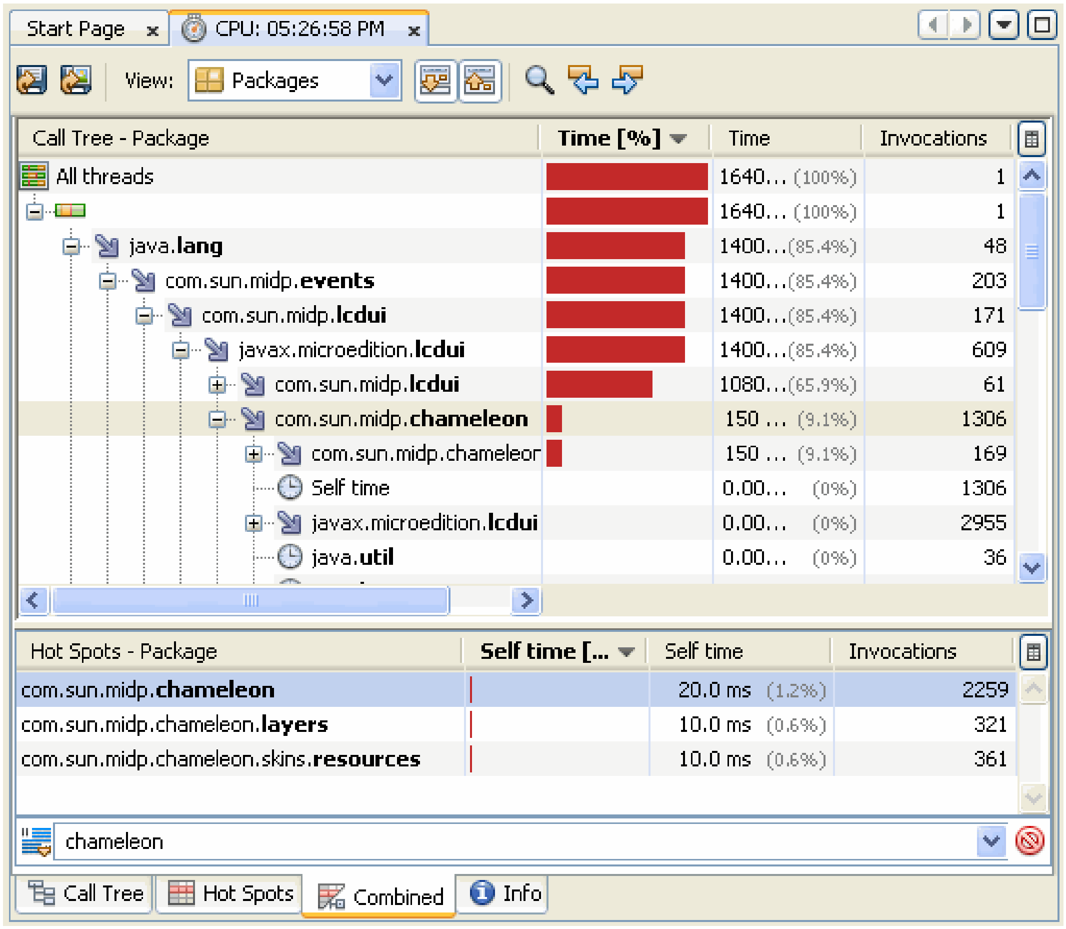
## Viewing Profiler Data

This procedure views data collected with the Profiler. See Saving Profiler Data. The data can be seen immediately after it is collected, as described in Saving Profiler Data, or when it is imported, as described in Loading Profiler Data.

---

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

---



Start Page | CPU: 05:26:58 PM

View: Packages

| Call Tree - Package | Time [%] | Time | Invocations |
|---|---|---|---|
| All threads | | 1640... (100%) | 1 |
| | | 1640... (100%) | 1 |
| java.**lang** | | 1400...(85.4%) | 48 |
| com.sun.midp.**events** | | 1400...(85.4%) | 203 |
| com.sun.midp.**lcdui** | | 1400...(85.4%) | 171 |
| javax.microedition.**lcdui** | | 1400...(85.4%) | 609 |
| com.sun.midp.**lcdui** | | 1080...(65.9%) | 61 |
| com.sun.midp.**chameleon** | | 150 ... (9.1%) | 1306 |
| com.sun.midp.chameleor | | 150 ... (9.1%) | 169 |
| Self time | | 0.00... (0%) | 1306 |
| javax.microedition.**lcdui** | | 0.00... (0%) | 2955 |
| java.**util** | | 0.00... (0%) | 36 |

| Hot Spots - Package | Self time [... | Self time | Invocations |
|---|---|---|---|
| com.sun.midp.**chameleon** | | 20.0 ms (1.2%) | 2259 |
| com.sun.midp.chameleon.**layers** | | 10.0 ms (0.6%) | 321 |
| com.sun.midp.chameleon.skins.**resources** | | 10.0 ms (0.6%) | 361 |

chameleon

Call Tree | Hot Spots | Combined | Info

You can change the granularity of the results. In the toolbar at the top of the tab, make a selection from the View menu.

**Method Level View** (default). Results are displayed according to the fully-qualified method names.

**Class Level View**. Results for all methods of the same class are aggregated in a single entry.

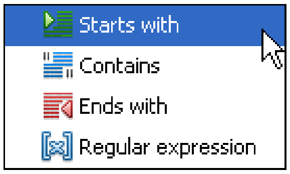
**Package Level View**. All methods of classes that belong to the same package are aggregated.

Click any column label to sort based on its values. To remove a column from the display, click the table icon above the scroll bar and uncheck the column name.

The above screenshot shows a view that combines the Call Tree and Hot Spots. Click the tabs along the bottom to see different views:

**Call Tree**. This tab displays a call tree showing the method call chain and the time/number of invocations for executing threads and methods in each context. (A context is a unique chain of method calls leading to the method's invocation.)

**Hot Spots**. This tab shows the total execution time and number of invocations for each method, irrespective of the context. It also includes the Method Name Filter Field, which filters based on the first column. To select the filtering parameters, click the filter icon and choose from the menu.



Enter a search string in the filter field. You can enter multiple values separated by spaces. To apply the filter, click the green check. To restore the unfiltered data, clear the filter field by clicking the red symbol.

**Combined**.This tab displays the Call Tree information in the upper part of the window and the Hot Spot data in the lower part.

**Info**. This tab displays the time the snapshot was taken, where it is saved, and the configuration of the profiling session.

You can right-click a profiling result to access additional utilities. The actions you see depend upon the currently selected view.

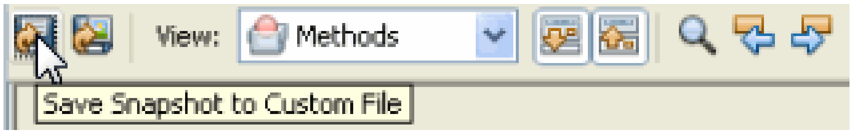
- **Show Subtree**. Displays the subtree for the selected method.
- **Show Back Traces**. Displays the back traces for the selected method.
- **Find in Hot Spots**. Displays the selected class or method in the Hot Spots tab.

See Saving Profiler Data, Loading Profiler Data and Saving Customized Snapshots and Images.

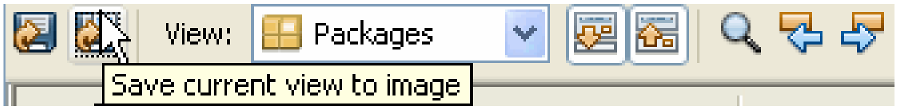
## Saving Customized Snapshots and Images

As you are Viewing Profiler Data you might isolate portions of the data you want to inspect later. You can take a snapshot of your current view at any time and import the snapshot later on.

Click the Save Snapshot to Custom File icon and specify a location for the snapshot file. The snapshot extension is `.nps` and the default location is `C:\Documents and Settings\`*`user`*`\My Documents`.



To create a bitmap image of the profile data, click the Save current view to image icon. The default format is `.png` (portable networks graphic file).



## Loading a Customized Snapshot

Follow these steps to load a customized profiler snapshot:

1. In the main window toolbar, select Tools >Load Profiler Snapshot...
2. Choose a customized snapshot (`.nps` file).

   The default location is: `C:\Documents and Settings\`*`user`*`\My Documents`.

   The snapshot opens in its own tab in the main window labeled CPU:*time*.

CHAPTER 10

# Monitoring Network Traffic

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).

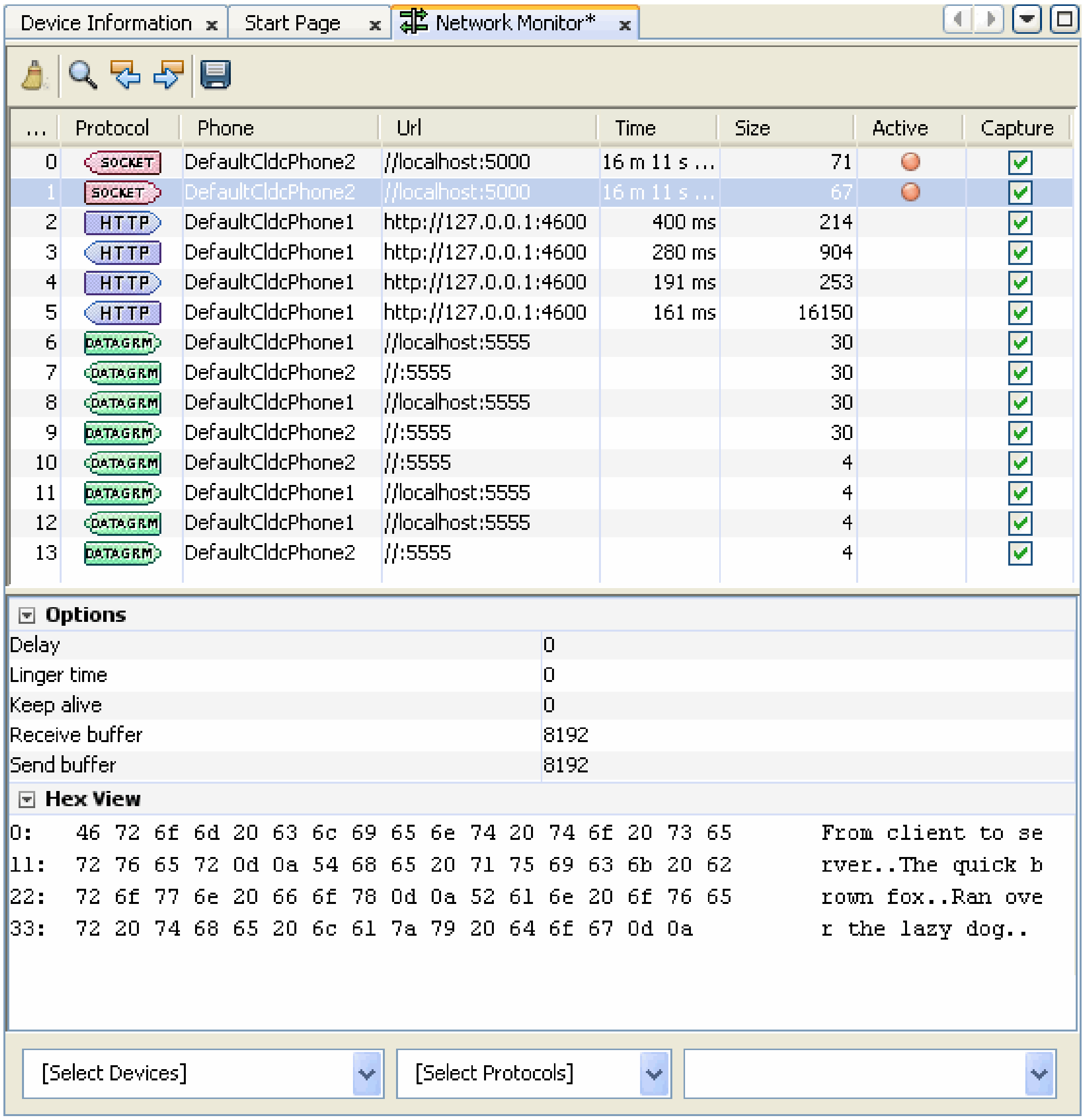
## Using the Network Monitor

Follow these steps to monitor network traffic:

1. In the Device Selector window, right-click on a device and choose Properties.
2. Check the Enable Network Monitor option. When you next run an application on the device, the network monitor opens.

   You can also edit the `device.properties` file to set `netmon.enabled` to `true`. The properties file is located in the device instance directory, as described in `/javame-sdk/3.0/work`.
3. Start your application.

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



Device Information | Start Page | Network Monitor*

| ... | Protocol | Phone | Url | Time | Size | Active | Capture |
|---|---|---|---|---|---|---|---|
| 0 | SOCKET | DefaultCldcPhone2 | //localhost:5000 | 16 m 11 s ... | 71 | ● | ✓ |
| 1 | SOCKET | DefaultCldcPhone2 | //localhost:5000 | 16 m 11 s ... | 67 | ● | ✓ |
| 2 | HTTP | DefaultCldcPhone1 | http://127.0.0.1:4600 | 400 ms | 214 | | ✓ |
| 3 | HTTP | DefaultCldcPhone1 | http://127.0.0.1:4600 | 280 ms | 904 | | ✓ |
| 4 | HTTP | DefaultCldcPhone1 | http://127.0.0.1:4600 | 191 ms | 253 | | ✓ |
| 5 | HTTP | DefaultCldcPhone1 | http://127.0.0.1:4600 | 161 ms | 16150 | | ✓ |
| 6 | DATAGRM | DefaultCldcPhone1 | //localhost:5555 | | 30 | | ✓ |
| 7 | DATAGRM | DefaultCldcPhone2 | //:5555 | | 30 | | ✓ |
| 8 | DATAGRM | DefaultCldcPhone1 | //localhost:5555 | | 30 | | ✓ |
| 9 | DATAGRM | DefaultCldcPhone2 | //:5555 | | 30 | | ✓ |
| 10 | DATAGRM | DefaultCldcPhone2 | //:5555 | | 4 | | ✓ |
| 11 | DATAGRM | DefaultCldcPhone1 | //localhost:5555 | | 4 | | ✓ |
| 12 | DATAGRM | DefaultCldcPhone1 | //localhost:5555 | | 4 | | ✓ |
| 13 | DATAGRM | DefaultCldcPhone2 | //:5555 | | 4 | | ✓ |

**Options**

| | |
|---|---|
| Delay | 0 |
| Linger time | 0 |
| Keep alive | 0 |
| Receive buffer | 8192 |
| Send buffer | 8192 |

**Hex View**

```
0:    46 72 6f 6d 20 63 6c 69 65 6e 74 20 74 6f 20 73 65      From client to se
11:   72 76 65 72 0d 0a 54 68 65 20 71 75 69 63 6b 20 62      rver..The quick b
22:   72 6f 77 6e 20 66 6f 78 0d 0a 52 61 6e 20 6f 76 65      rown fox..Ran ove
33:   72 20 74 68 65 20 6c 61 7a 79 20 64 6f 67 0d 0a         r the lazy dog..
```

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.

---

**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.

---

See also: Filtering Messages, Sorting Messages, Saving and Loading Network Monitor Information, Clearing the Message Tree

## Filtering 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 supported protocols are datagram, socket, http, and https.
- Click the magnifying glass in the Network Monitor toolbar to search for a specific string in the data in the Phone or URL columns.

See also: Using the Network Monitor, Sorting Messages, Saving and Loading Network Monitor Information, Clearing the Message Tree

## Sorting Messages

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

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

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

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

---

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

---

See also: Using the Network Monitor, Filtering Messages, Saving and Loading Network Monitor Information, Clearing the Message Tree

## Saving and Loading Network Monitor Information

To save your network monitor session, click the disk icon in the Network Monitor toolbar.

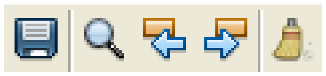
Choose a file name. This file name will be used for all the network monitor saves you make in this Java ME SDK session. The default directory is `C:\Documents and Settings\`*User*`\My Documents\` and the default file extension is `.nmd` (network monitor data).

To load a network monitor session, choose Tools > Load Network Monitor Snapshot... and browse to the data you saved.

See also: Using the Network Monitor, Filtering Messages, Sorting Messages, Clearing the Message Tree

## Clearing the Message Tree

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



See also: Using the Network Monitor, Filtering Messages, Sorting Messages, Saving and Loading Network Monitor Information

CHAPTER 11

# 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.

LWUIT is an open source project at `https://lwuit.dev.java.net/`. For more information see the Lightweight UI Toolkit Developer's Guide.

LWUIT supports the following resource elements: images, animation, bitmap fonts, localization bundles, and themes. Resources are delivered as a bundle - a binary file that can be loaded and used on the device. Java ME Platform SDK supports LWUIT with an integrated Resource Manager for creating and maintaining resource bundles.

- Adding the LWUIT Library
- Using the LWUIT Resource Manager

## Adding the LWUIT Library

The LWUIT library can be added to any MIDP project.

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, select LWUIT and click Add Library.

   You can see the package under Libraries and Resources.

# Using the LWUIT Resource Manager

The Resource Manager is a graphical tool for creating resource bundles and adding them to the build process.

1. Select a project that contains the LWUIT libraries.
2. Select Tools > LWUIT Resources.

   The Resource Manager opens.
3. To add a bundle, right-click anywhere within the Resource Manager window and select Add Bundle from the context menu.
4. Enter a bundle name and click OK.

   You are ready to add resources to the new bundle.
5. Select a resource, and select Windows > Properties to view and edit the resource.

Experiment with adding resources. Resources are fully described in the Lightweight UI Toolkit Developer's Guide. Here is a brief summary of the resource types:

**JPG and PNG files**. The file name can be changed and you can choose to pack the file to save space.

**Animations**. GIF files. You just supply a name.

**Font**. There are two font types:

**System (Create New)**.

Created from a system font. You can choose the system font in properties. Available fonts are: Dialog, DialogInput, Monospaced, Serif, and SansSerif.

- **From File (Create from File)**.

  Choose the file from which the font resource will be created. You can edit the font's name, boldness, the size of the font, and whether or not antialiasing is used.

**Localization**. Choose the main localization bundle, for example, `foobar.properties`. This main bundle is then added with a "default" ID. Other locales are added with their proper ID. For example, `foobar_en_GB.properties` is added with the ID `en_GB`. Unfortunately this resource must be recreated in the resource manager when more locales are added (or removed).

**Theme**. Adds the `.conf` theme file. See the Lightweight UI Toolkit Developer's Guide.

CHAPTER 12

# Security and MIDlet Signing

The Java ME Platform SDK supports the security policies and domains defined by both JSR 185 (Java Technology for the Wireless Industry or JTWI) and JSR 248 (Mobile Service Architecture or MSA). The SDK provides tools to sign MIDlet suites, manage keys, and manage root certificates. The security domains are further described in Security Domains.

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 Java ME Platform 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:

- Security Domains
- Setting Security Domains
- Signing a Project
- Managing Keystores and Key Pairs
- Managing Root Certificates

## Security Domains

The SDK supports five security domains for MSA:

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

`identified_third_party`. Intended for MIDlets whose origins were determined using cryptographic certificates. Permissions are not granted automatically, but the user is prompted less often than for the `unidentified_third_party` domain.

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

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

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

The SDK includes four JTWI security domains:

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

`trusted` - All permissions are granted to MIDlets in this domain.

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

`maximum` - All permissions are granted to MIDlets in this domain (equivalent to `trusted`.) Maximum is the default value.

## Setting Security Domains

In the SDK, when you use Run 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 suite is not signed, it is placed in the default security domain.
- If the MIDlet is signed, it is placed in the protection domain that is associated with the root certificate of the signing key's certificate chain. See "Signing a Project" on page 95.

### Specifying the Security Domain for an Emulator

To choose the security domain you want the emulator to use, right-click on the device in the Device Selection window and select a security domain. The SDK knows the runtimes the device can support and supplies only possible domains. The default for both MSA and JTWI is Maximum. See Setting Device Properties.

### Specifying the Security Domain for a Project

You can set a security domain in a project. Follow these steps to specify the project security domain:

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.

## 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 key pair consists of two keys:

- 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 Managing Keystores and Key Pairs.

Follow these steps to sign a project with a signing 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 Creating a Keystore.
6. Click OK.

---

**Note –** CDC projects cannot be signed with this tool. 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
```

---

See Signing.

## 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 that can be used by anyone 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:

- Security Domains
- Creating a New Key Pair
- Removing a Key Pair
- Importing an Existing Key Pair

You can also use the command line tools described in Command Line Security Features.

## 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 Signing.

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

See Creating a Keystore, Adding an Existing Keystore, Creating a New Key Pair, Removing a Key Pair and Importing an Existing Key Pair

### Creating a Keystore

Follow these steps to create a new keystore.

1. Select Tools > Keystores.

   The Keystores Manager opens.

2. Click Add Keystore.

   The Add Keystores window opens.

3. Choose Create Keystore.

4. Supply a name, location, and password.

   The default location is C:\Documents and Settings\*User*.

5. Click OK.

   The new keystore appears in the Keystores list.

## Adding an Existing Keystore

You can make an existing keystore available to Java ME Platform SDK.

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. Click OK.

   The default location is C:\Documents and Settings\*User*.

5. Click OK.

   The new keystore appears in the Keystores list.

## Creating 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, select Add Keystore to create a new keystore or add an existing keystore.

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"

The password is optional. If you do provide a password, it must have at least six characters.

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 "Signing a Project" on page 95.

### Removing a Key Pair

1. Select Tools > Keystores.
2. In the Keys area, click on a Key Pair.
3. Select Delete.

### Importing 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 Keystores.

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

## Managing Root Certificates

The Java ME Platform SDK command line tools described in Managing 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 *users.home*`/javme-sdk/3.0/work/`*emulator-instance*`/appdb`.

The micro keystore, `_main.mks` resides in the following directory.

*installdir*`\runtimes\cldc-hi\appdb\_main.mks`

This directory also contains `keystore.ks` and `serverkeystore.ks`. You can use the `-import` option to import certificates from these keystores as described in Managing Certificates (MEKeyTool).

CHAPTER 13

# BD-J Support

The Java ME Platform SDK ships with a JAR file containing stubs for the BD-J platform. The stubs allow you to compile a CDC project as a Blu-ray Disc Java (BD-J) application. Once built, the application can be burned to a Blu-ray disc and played on a Blu-ray disc player, or, it can be played online with a full-featured player such as TotalMedia Theatre, WinDVD, or PowerDVD.

The SDK supports TotalMedia Theatre. If it is installed, running a BD-J project launches your application in the TotalMedia Theatre Player.

To obtain source and Javdocs for the stubs, contact the Blu-ray Disc Association (`http://www.blu-raydisc.com`). You can email license requests to License@bdamail.com.

See the following topics:

- Creating a Stubs for BD-J Platform Project
- Compiling, Deploying, and Running a Stubs for BD-J Platform Project

## Creating a Stubs for BD-J Platform Project

Follow these steps to create a BD-J Project.

1. Select File > New Project, and choose CDC Application. Click Next.
2. Set the project name, location, and Main class name and click Next.
3. Select Platform. From the Java Platform menu, choose Stubs for BD-J Platform and click Finish.

   A simple BD-J hello World project is created.

4. To set BD-J preferences and permissions, right-click on the project and choose Properties. In the Category area select Application Descriptor. Click the BD-J tab on the right.

   The properties in this window are used to create a Permission Request File (PRF). This is an XML file and its format is controlled by Blu-ray disc specifications. Typically permissions apply to signed applications. Unsigned applications are not allowed access.

   - **Application ID** is a required value. The default is generally acceptable. This value is a 16-bit hexadecimal string. The recommended range is 0x0000 to 0x3FFF for unsigned applications, and 0x4000 to 0x7FFF for signed applications.
   - **Organization ID** is a required value. The default is generally acceptable, but it can be changed. This value is a 32-bit hexadecimal string. The recommended range is 0x7FFF0001 to 0x7FFFFFFF.
   - The SDK uses the Service Access, User Settings and Preferences, and Network Permissions values to create a permissions file.
   - **File Access** - Allows a signed application to access persistent storage.
   - **Lifecycle Control** - Allows an application to control the lifecycle (stop, pause, or resume) of any application that it has launched. It cannot affect applications it has not launched.
   - **Service Selection** - Allows a signed application to select a new service (assuming other permissions also permit the selection).
   - **User Settings and Preferences** - Checking Read or Write requests permission for all preferences. The user preferences are: User Language, Parental Rating, DefaultFontSize, and Country Code. For example, check Read to enable Read permission for all preferences.
   - **Network Permissions** - Allows a signed application to specify the hosts and actions that receive the permissions.

     For example, enter `connect = *` to request permission for all hosts to connect. To open all permissions for all hosts, enter:
     `connect, listen, resolve, accept = *`
   - By default the deployment directory `${build.dir}/deploy` is created in:

     `C:\Documents and Settings\`*User*`\My Documents\`
     `JavaMESDKProjects\`*Application*`\build\deploy`

     The deployment directory contains the JAR file.

The PRF is stored in the JAR file in the same location as the Main class. It is given the name `bluray.`*Main*`.perm`, where *Main* is the name of the Main class. A PRF file might look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<n:permissionrequestfile xmlns:n="urn:BDA:bdmv;PRF" orgid=
"56789abc" appid="00004001">
  <file value="true"></file>
  <applifecyclecontrol value="false"></applifecyclecontrol>
  <servicesel value="true"></servicesel>
  <userpreferences read="true" write="false"></userpreferences>
  <network>
    <host action="connect, listen, resolve, accept"> * </host>
</n:permissionrequestfile>
  </network>
```

The contents of the deployment directory can be run automatically if you specified a valid player when you created the External BD-J Platform.

See Compiling, Deploying, and Running a Stubs for BD-J Platform Project.

## Compiling, Deploying, and Running a Stubs for BD-J Platform Project

To deploy a project (see Creating a Stubs for BD-J Platform Project), right-click the project and select Compile or Deploy. The default deployment location is:

`C:\Documents and Settings\`*User*`\My Documents\JavaMESDKProjects\`*Application*`\build\deploy`.

To run the project, right-click on the project name and select Run. If you have ArcSoft TotalMedia Theatre installed, running the project launches the application in TotalMediaTheatre. If you specified an alternate player when you created the External BD-J platform, the SDK will attempt to open the deployment directory with that executable.

When the directory is populated, use a player that supports BDMV to open the deployment directory (for example, WinDVD, PowerDVD or TotalMedia Theatre).

To create a valid playable disc, burn the deployment directory on a Blu-ray disc. The disc should play on a Blu-ray player.

CHAPTER 14

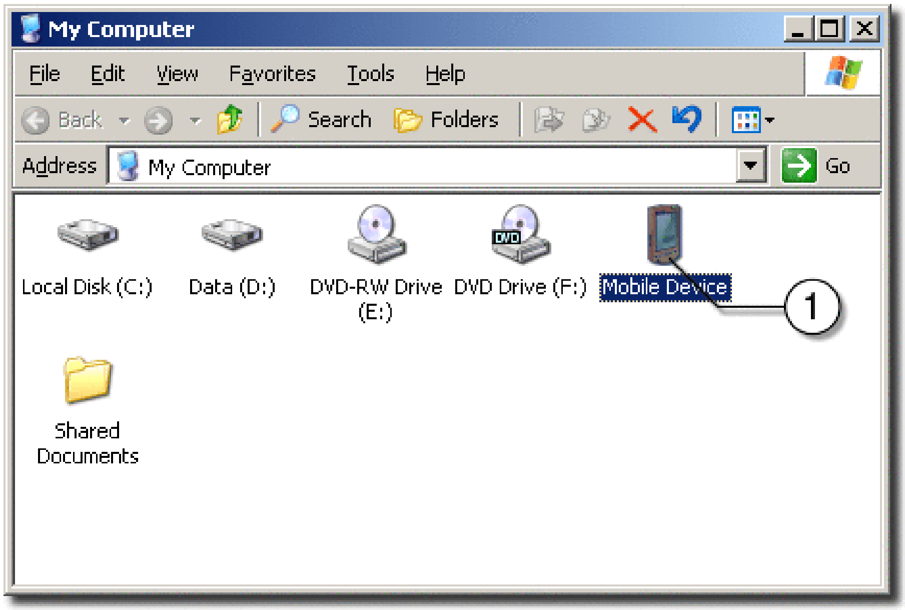
# CLDC Emulation on a Windows Mobile Device

This procedure describes how to install Sun Java CLDC Emulation software on a real device and make it available to the Java ME Platform SDK software on the host computer. It also provides a procedure for testing on-device debugging.

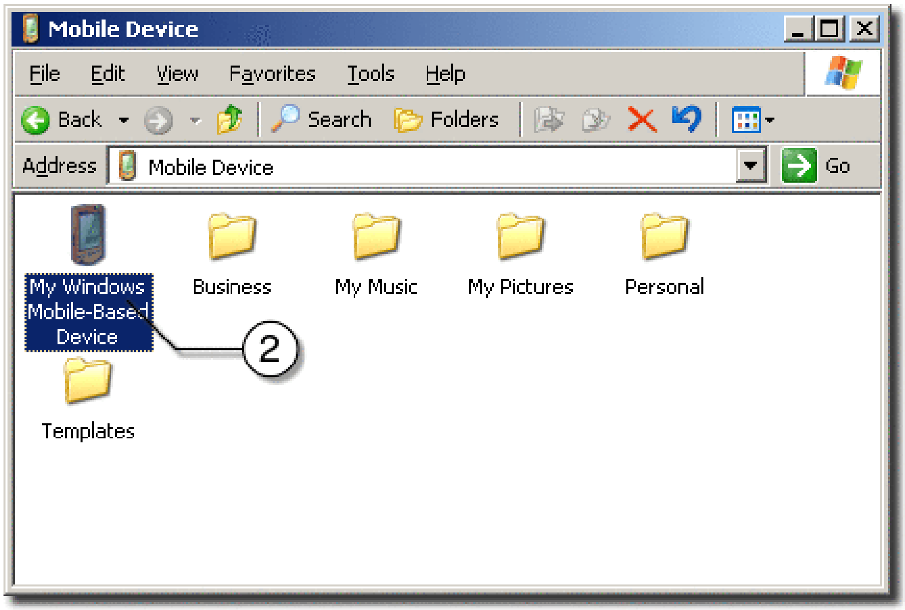
- CLDC Emulator Installation for a Device Running Windows Mobile
- Testing On-device Debugging

## CLDC Emulator Installation for a Device Running Windows Mobile

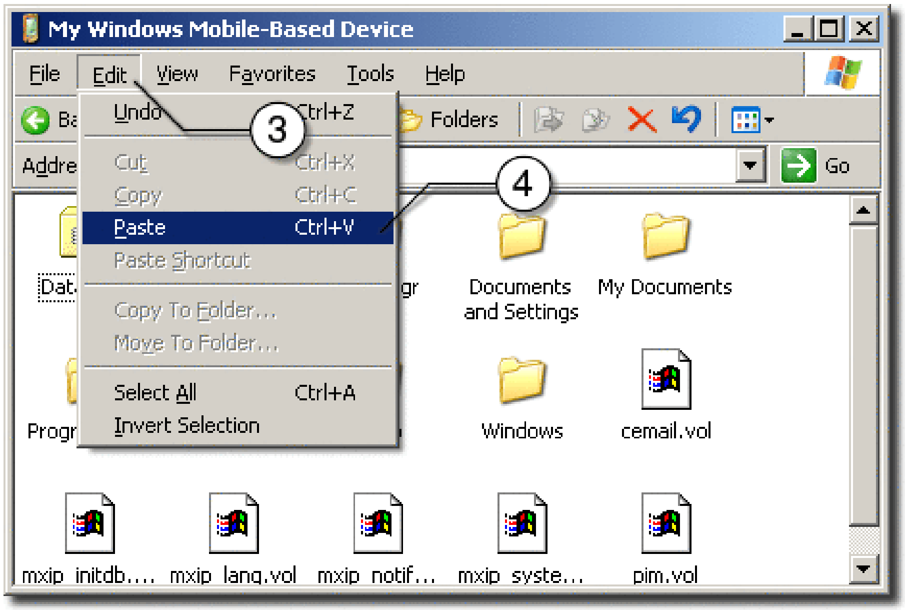
1. Connect the device to your host computer and register it with `ActiveSync`.
2. Copy the Sun Java CLDC Emulation Installation CAB file onto the clipboard using Windows Explorer.
   a. Browse to *JavaMESdkHome*`\emulator-dev-install`.
   b. Right-click on `sun-java-cldc-emu.cab`.
   c. Select Copy from the context menu.
3. Paste the CAB file into the device root directory.
   a. In Windows Explorer, open Mobile Device.



b. Open My Windows Mobile-Based Device.



c. Open the Edit menu.

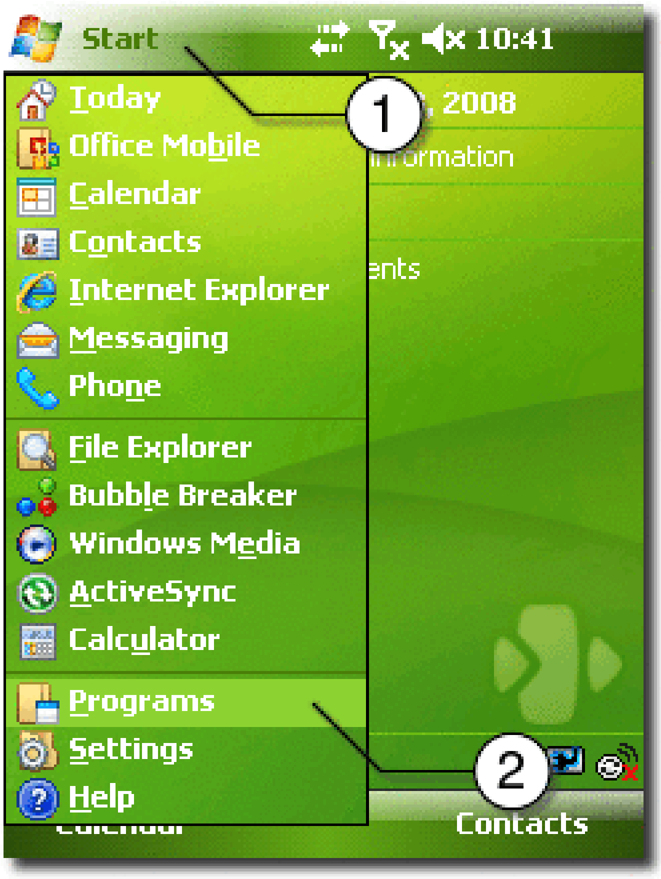


d. Click Paste to insert the CAB file.

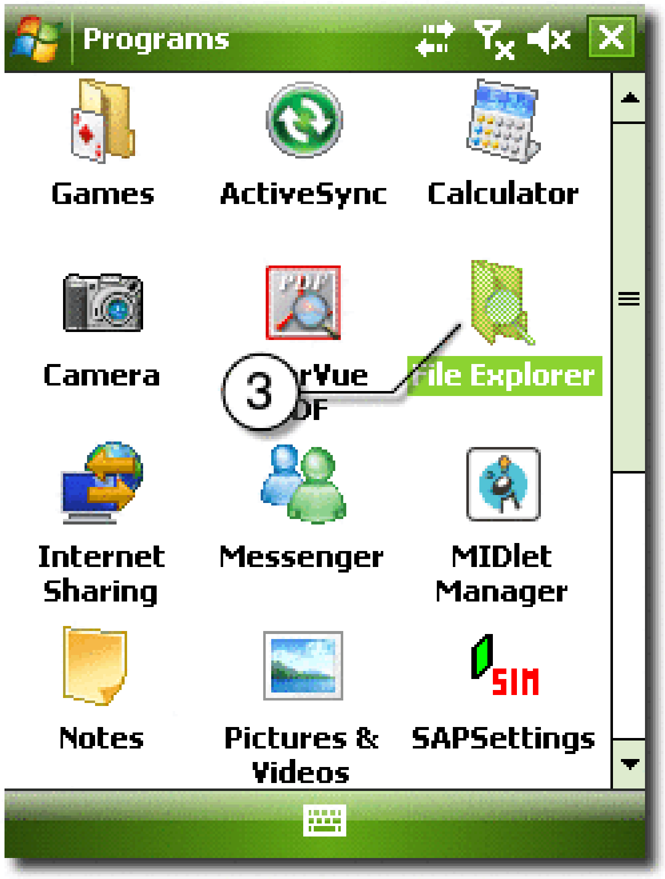
4. Run the File Explorer on your device.

   a. Open the Start menu.

   b. Click Programs.



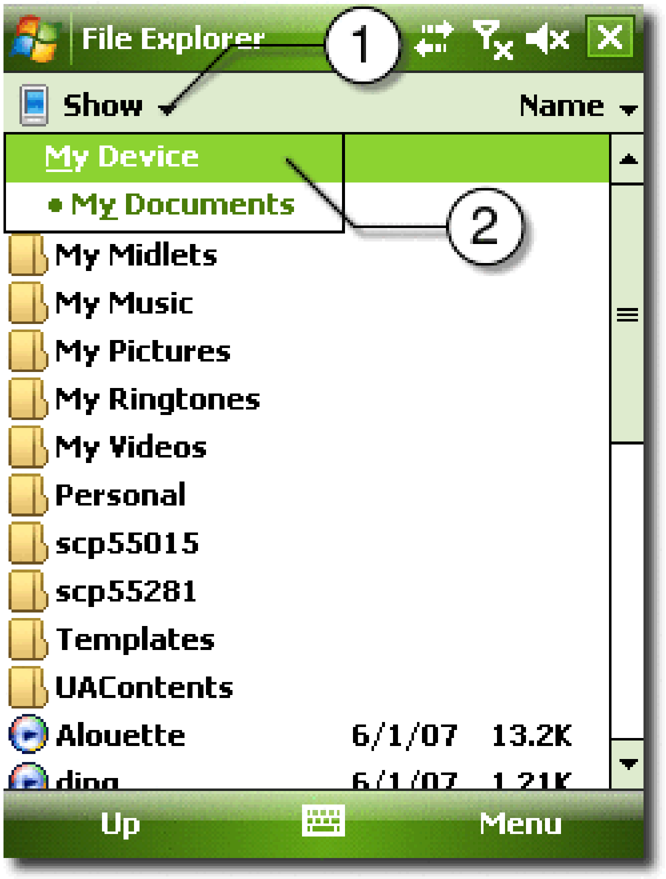
c. Click File Explorer.



5. Start the CAB installation on the device.

   a. Open the Show menu.

   b. Select My Device.

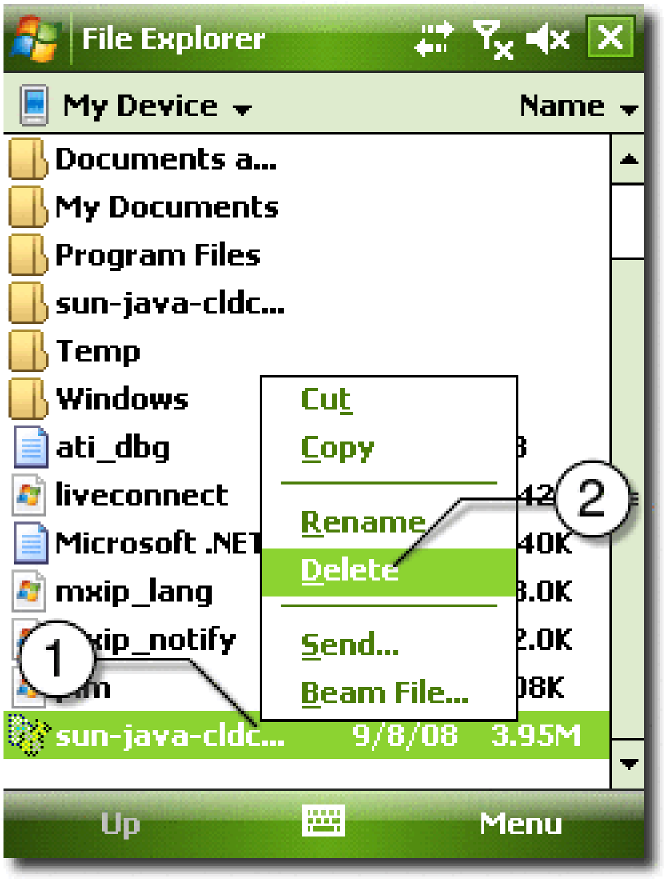


c. Click on the `sun-java-cldc-emu.cab` file.

6. If asked during the installation, install the application on the device.
7. Wait for the installation to finish.
8. You can delete the CAB file after the installation is complete.

   a. Press on sun-java-cldc-emu.cab label until the context menu opens.

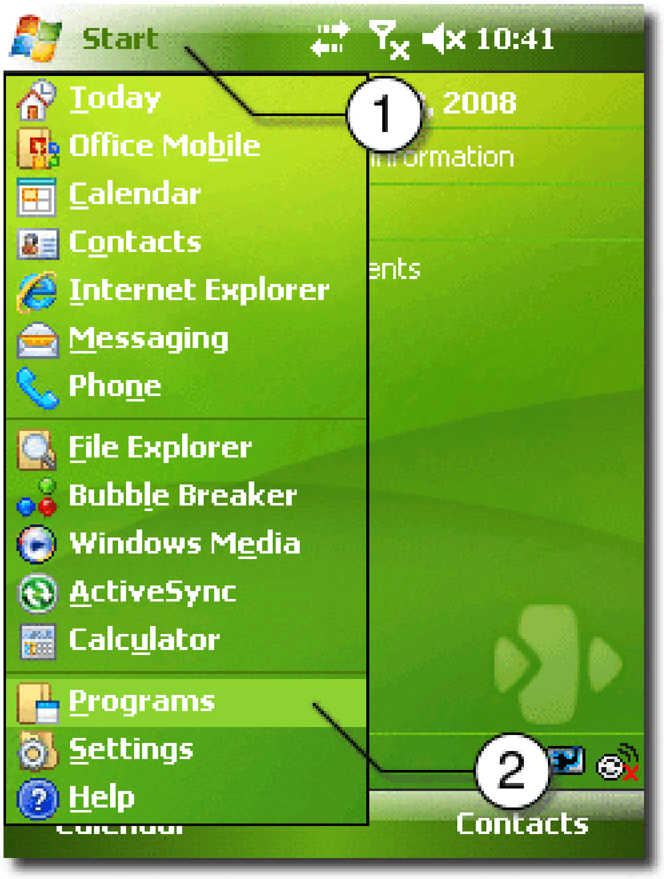
   b. Click Delete.



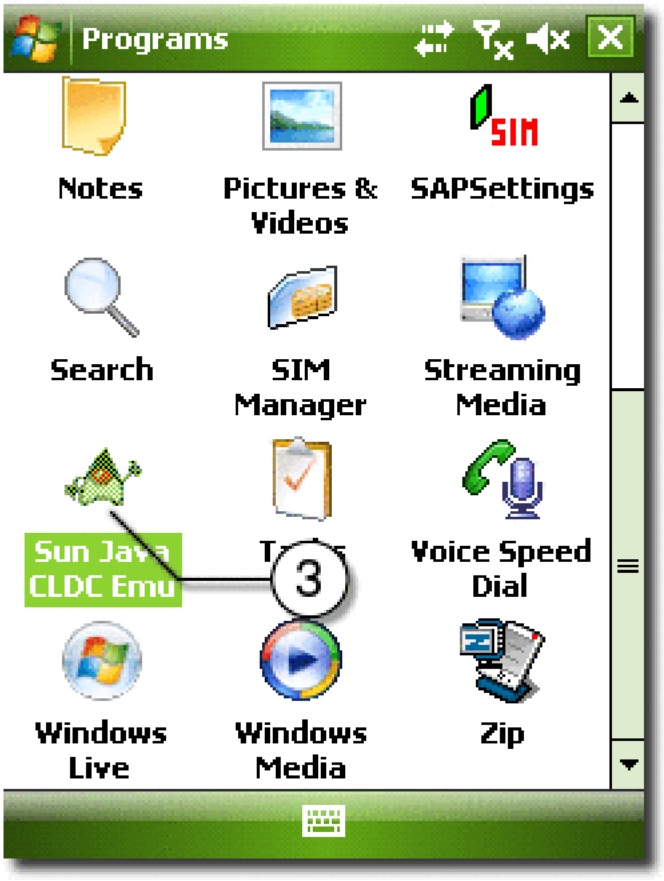
9. Run the Sun Java CLDC Emulation on the device.

   a. Open Start menu.

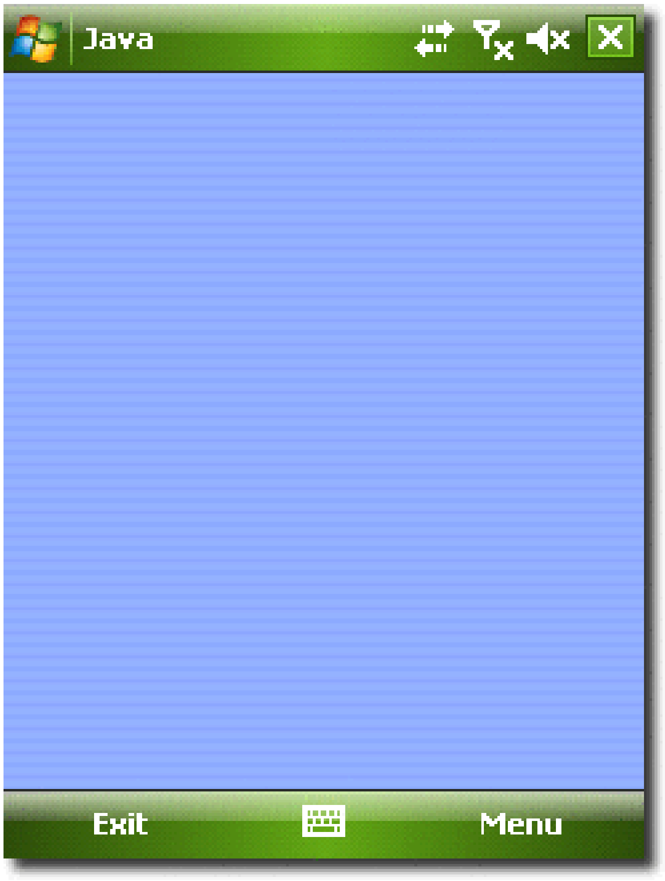
   b. Click Programs.



c. Click Sun Java CLDC EMU.



10. Wait for the Sun Java CLDC Emulation to start.



11. Allow up to 30 seconds (the default value) for Java ME SDK to recognize the connected device and the Sun Java CLDC emulation software.

    When the device is recognized a new device, CldcWinceEmu*number* (for example, CldcWinceEmu1) should appear in the Device Selector window, and the output from the command `emulator.exe -Xquery` should also be displayed.

    You can select this device as a target device in the user interface, or if you run the emulator from the command line it can be used as an argument. For example:

    `emulator -Xdevice:CldcWinceEmu1` ...

See Testing On-device Debugging.

## Testing On-device Debugging

This procedure provides instructions for running the FPDemo sample project on a device running Windows Mobile.

Before starting this procedure:

- Integrate the device as described in CLDC Emulator Installation for a Device Running Windows Mobile.
- Confirm the device is connected and that it appears in the Device Selector window.
- If the output console is not visible, select Window > Output > Output to open it.

This procedure features command line debugging, but you can use a graphical debugger in a similar fashion.

1. In the Java ME Platform SDK, select File > Open Sample Project > FPDemo.
2. In the Projects window, right-click on FPDemo and select Properties.
3. Choose the Platform category. From the Device drop-down menu, select the name of the connected device. Click OK.
4. Select FPDemo and select Run > Run in Debug Mode, or click the corresponding icon on the toolbar.

   The application is now deployed and started on the connected device.

   Note the Port number displayed in the Output console.
5. Open a Windows shell.
6. Start `jdb` with the following command:

   ```
   db -sourcepath installdir\apps\FPDemo\src -connect
   com.sun.jdi.SocketAttach:hostname=127.0.0.1,port=port-number
   ```
7. Set the breakpoint at the place where the demo handles its menu commands:

   ```
   stop in calculator.CalculatorMIDlet.commandAction
   ```
8. In the FPDemo application, enter two numbers, choose an operation, and invoke the Calc command.

   The debugger displays the place where the execution stops.
9. To step through the source past the point where the result is calculated, use the `next` command until you get past the following line:

   ```
   res = n1 <op> n2;
   ```

10. Check that the input and the calculated values are correct:

    ```
    eval n1, eval n2, eval res
    ```

11. Override the calculated value as follows:

    `set res =` *new-value*

12. Clear the breakpoint as follows:

    ```
    clear calculator.CalculatorMIDlet.commandAction
    ```

13. Let the application continue:

    ```
    cont
    ```

    You can see that the application displays the overridden result.

14. Exit the debugger:

    ```
    exit
    ```

15. Exit FPDemo.

CHAPTER **15**

# Installing CLDC Emulation on a Windows Mobile Emulator

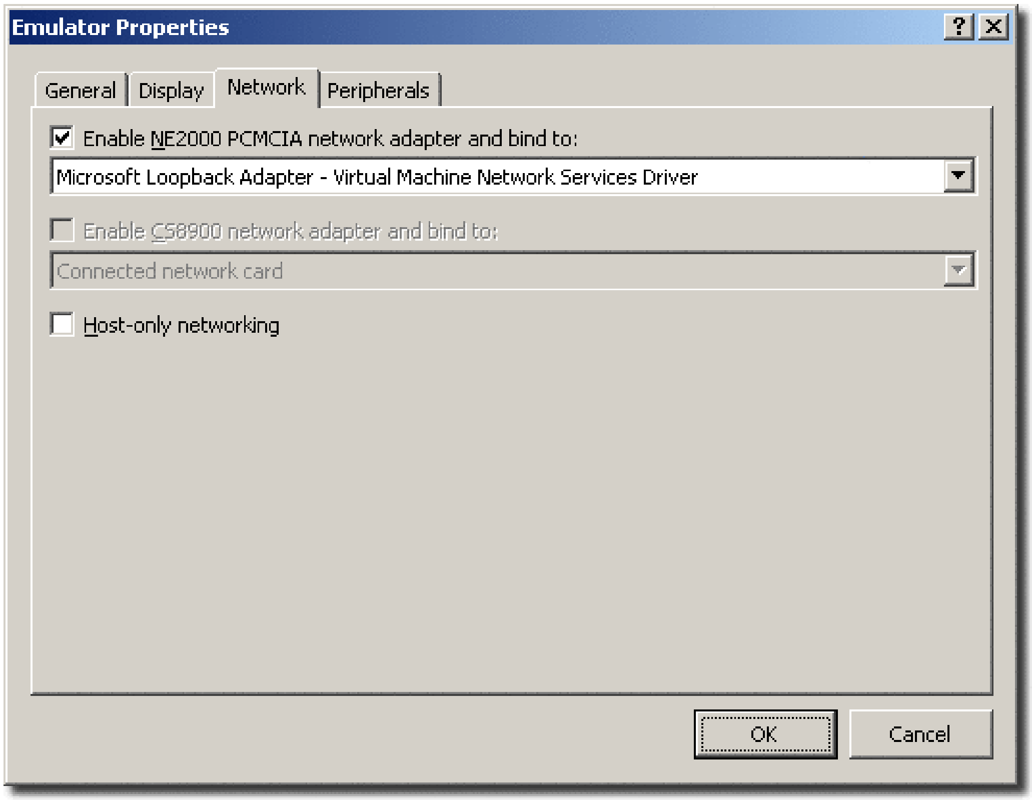
This procedure describes how to install the Sun Java CLDC platform binary for the ARM processor into the Windows Mobile Emulator.

## CLDC Installation for Windows Mobile

1. Download and install Microsoft Device Emulator with device images for Windows Mobile 6.
2. Download and install Microsoft Virtual PC.

**Note –** Installs virtual switch driver required for emulated network adapters.

3. Run Windows Mobile 6 Professional Emulator and bind its emulated NE2000 Network Adapter to a desktop network card.


Emulator Properties

☑ Enable NE2000 PCMCIA network adapter and bind to:
Microsoft Loopback Adapter - Virtual Machine Network Services Driver

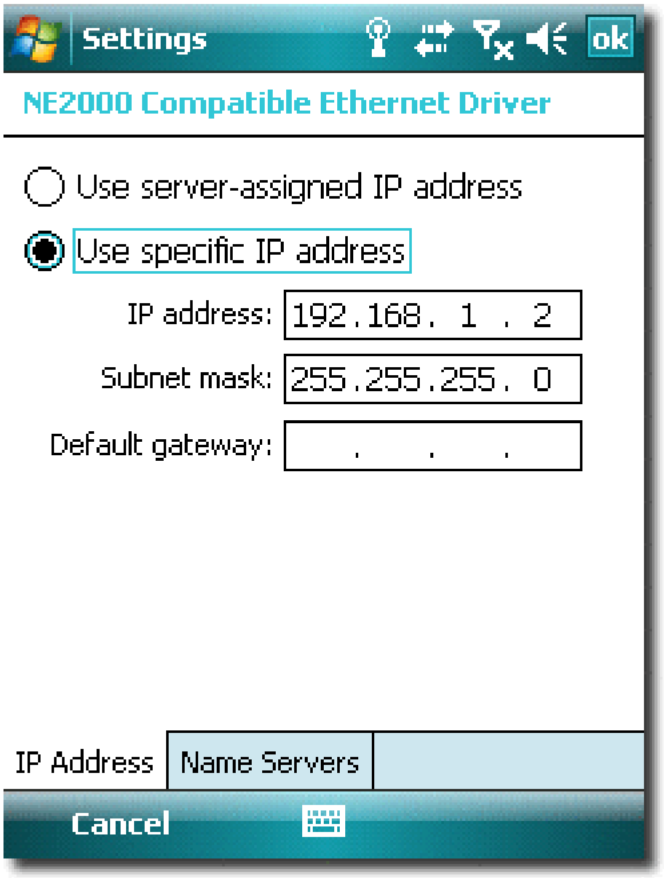
☐ Enable CS8900 network adapter and bind to:
Connected network card

☐ Host-only networking

- Consult the Microsoft device emulator documentation for instructions.
- Don't use ActiveSync for networking (cradling the emulator).

4. Write down the IP address of the emulator.

   Open the Start menu on the emulator, click Settings, click Connections, click Network Cards, Click NE2000 Compatible Ethernet Driver.



NE2000 Compatible Ethernet Driver

Use server-assigned IP address [✓ Use specific IP address]

IP address: 192.168. 1 . 2

Subnet mask: 255.255.255. 0

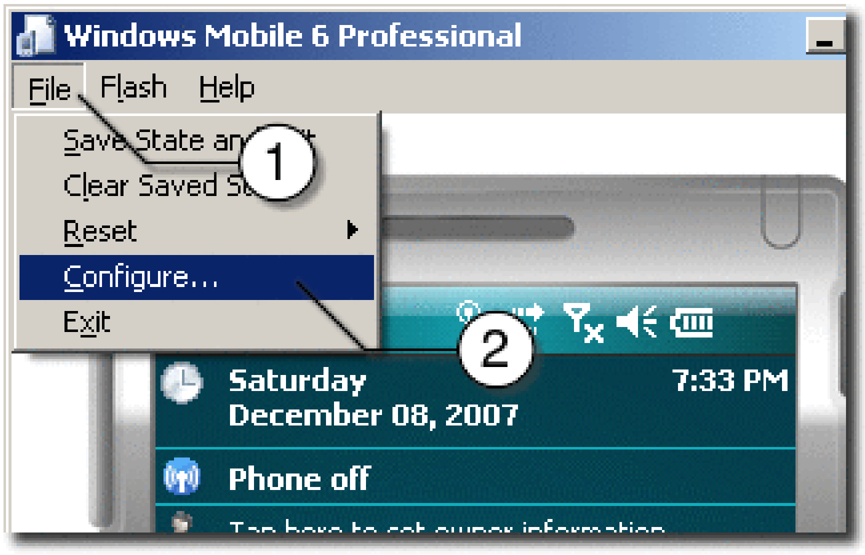
Default gateway: . . .

IP Address | Name Servers

5. Open Emulator Properties

   a. Open the File menu.

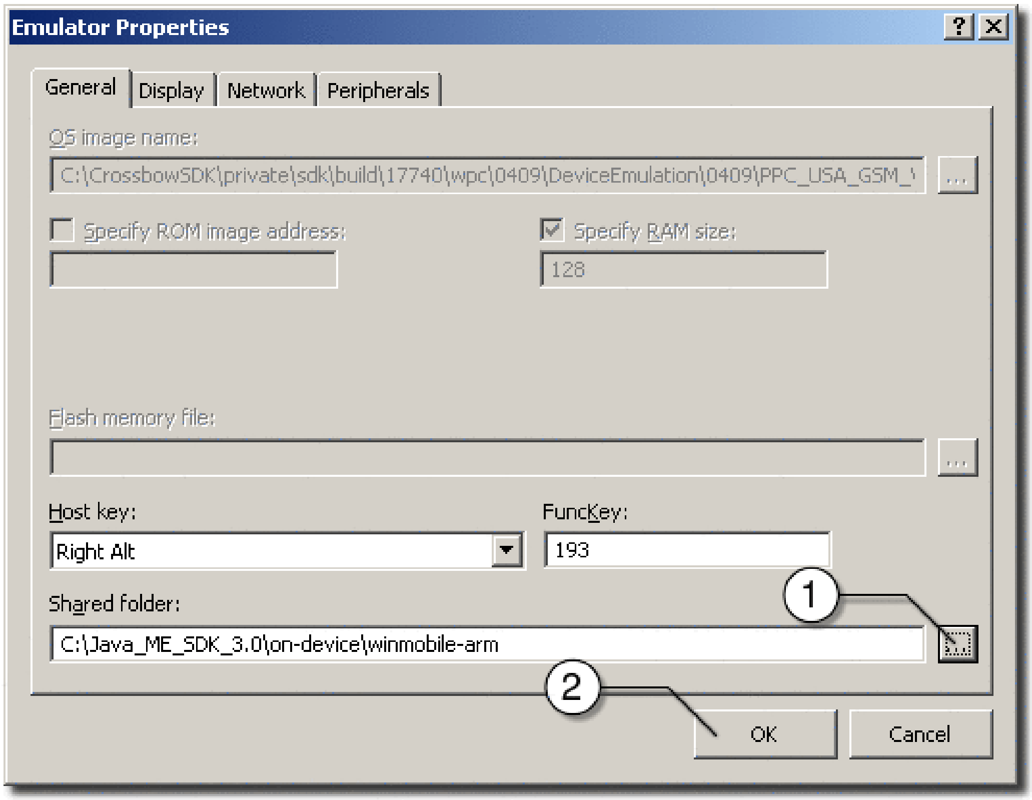
   b. Click Configure.



6. Set *JavaMESdkHome*`\on-device\winmobile-arm` as a shared folder.

   a. Browse to *JavaMESdkHome*`\on-device\winmobile-arm`.

   b. Click OK.



OS image name: C:\CrossbowSDK\private\sdk\build\17740\wpc\0409\DeviceEmulation\0409\PPC_USA_GSM_\

Specify ROM image address:

Specify RAM size: 128

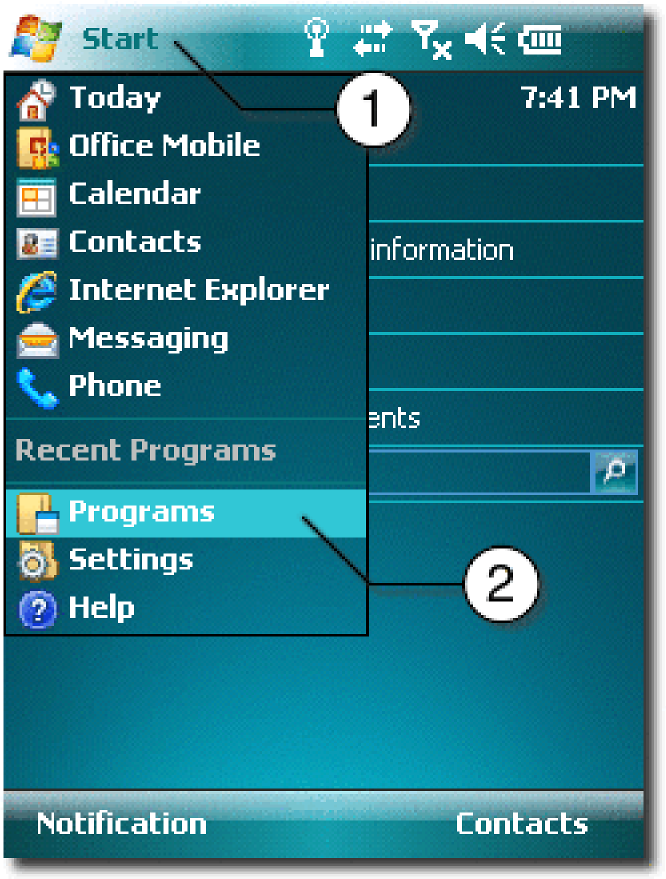
Flash memory file:

Host key: Right Alt

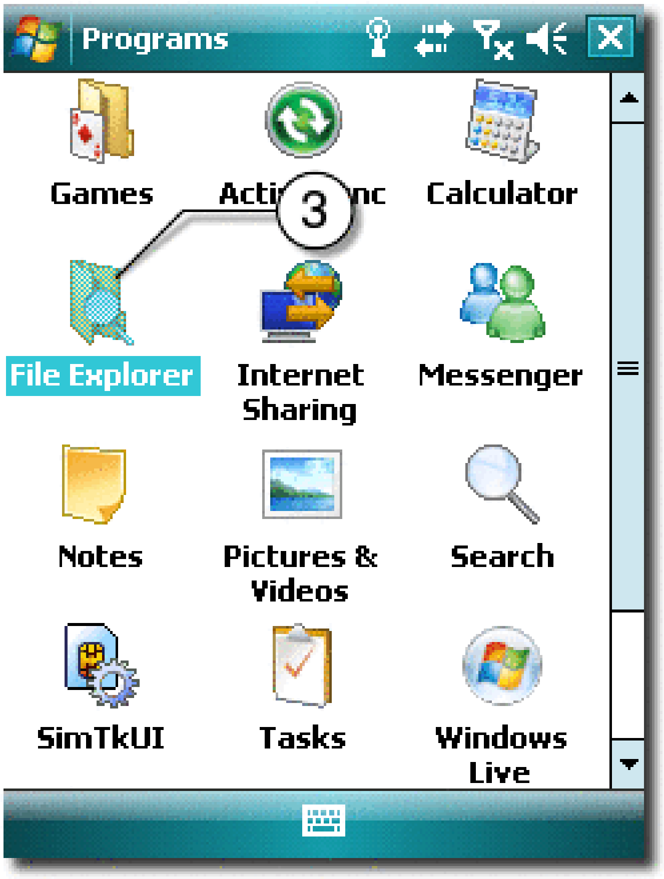
FuncKey: 193

Shared folder: C:\Java_ME_SDK_3.0\on-device\winmobile-arm

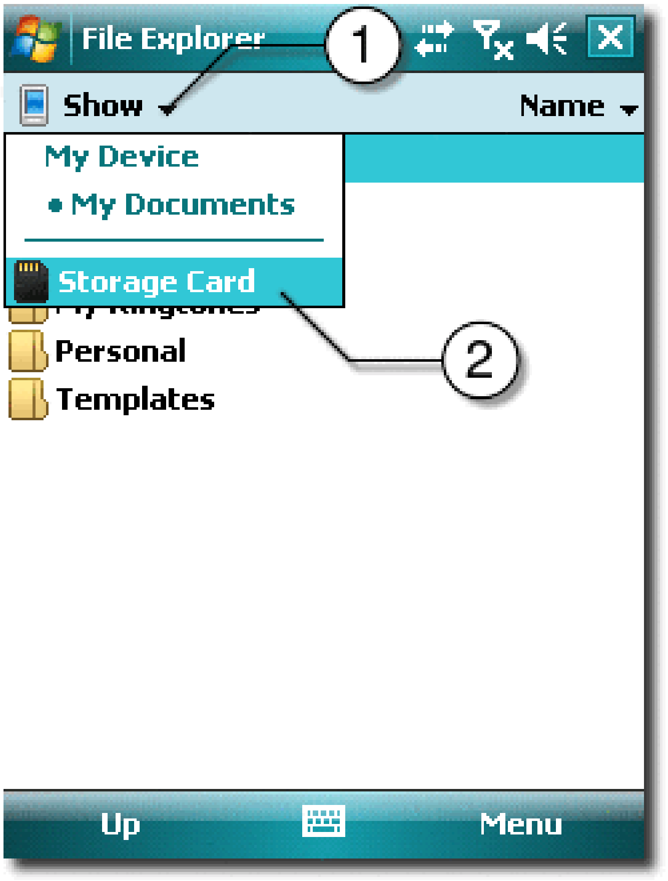
7. Run File Explorer on the emulator.
   a. Open Start menu.
   b. Click Programs.



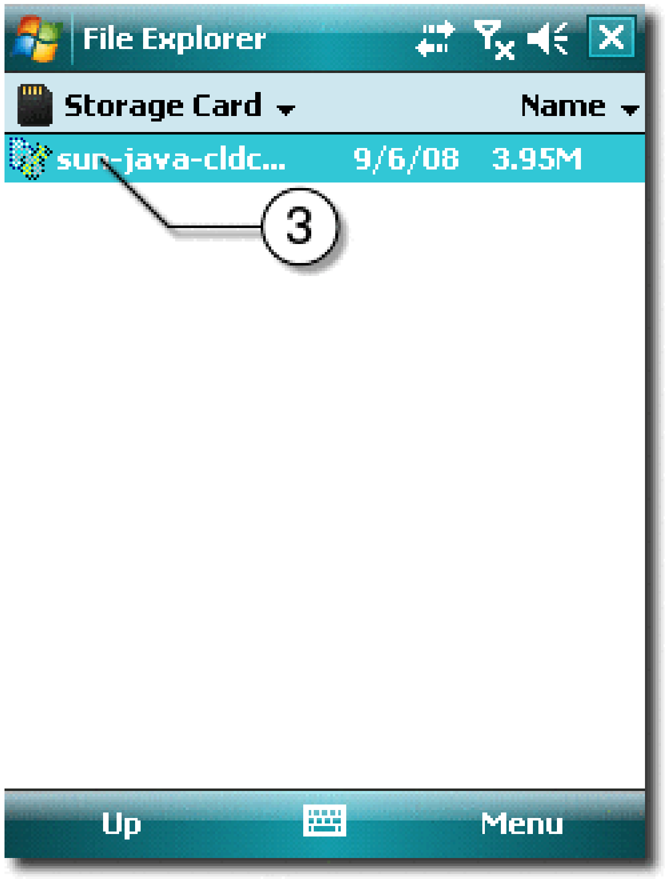
c. Click File Explorer.



8. Start the Sun Java CLDC Emulation CAB file installation.
   a. Open Show menu.
   b. Select Storage Card.



c. Click on the `sun-java-cldc-emu.cab` file.

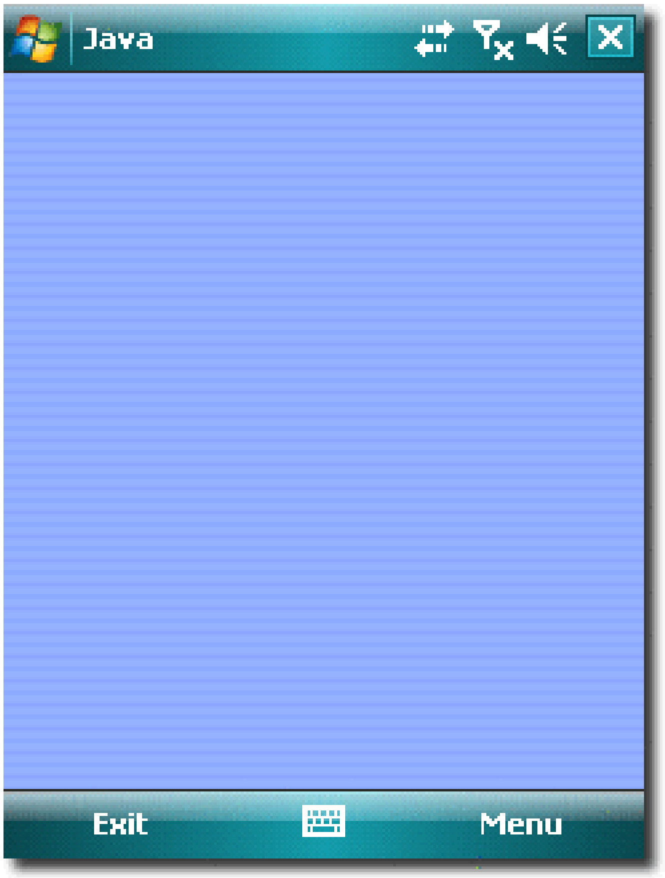


9. Finish the installation and run the Sun Java CLDC Emulation.

---

**Note –** See CLDC Emulator Installation for a Device Running Windows Mobile

---



10. Use the *installdir*`/bin/device-address` tool to register the emulator IP address in the SDK. See Managing Device Addresses (device-address).

    a. Execute the following command to ensure the SDK registers the emulator:

       `device-address.exe add ip` *address*

       *address* should be the IP address written down in step 4.

    b. After the device registers the emulator the device selector window should display the device as CldcWinceEmu*number* and it should also appear in the output when you issue the `emulator.exe -Xquery` command.

CHAPTER 16

# On-device Debugging

This section discusses the on-device debugging process for CLDC and CDC.When a project is run in debug mode, a graphical debugger or a command line debugger can be attached so that you can set and remove breakpoints.

See On-device Debugging Procedure and Sample CLDC Debugging Session.

## On-device Debugging Procedure

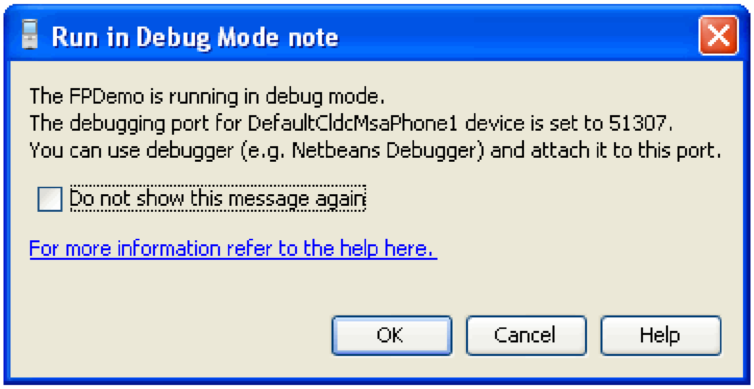
Before starting this procedure ensure your environment is properly configured:

- Integrate the device as described in CLDC Emulator Installation for a Device Running Windows Mobile.
- Confirm the device is connected and that it appears in the Device Selector window.
- If the output console is not visible, select Window > Output > Output to open it.
- If the project you want to debug is not main, right-click and select Set as Main Project.
- Only one debug process can run on a port. In the device selector, right-click on a device and look for the Debug Port property. By default the port number is 51307. If you are debugging on 51307 and you wish to debug an additional project, you must run on a different device and change its port number property.

This procedure features command line debugging, but you can use a graphical debugger in a similar fashion.

1. Select the project you want to debug and select Run > Run Project in Debug Mode or click the Run Main Project in Debug mode icon on the toolbar.

   The first time you debug you see this dialog.


Run in Debug Mode note

The FPDemo is running in debug mode.
The debugging port for DefaultCldcMsaPhone1 device is set to 51307.
You can use debugger (e.g. Netbeans Debugger) and attach it to this port.

☐ Do not show this message again

For more information refer to the help here.

2. Make note of the debugging port for the device, and click OK. If you suppress this dialog you can still get the port number from the Output window. Look for the message “`Starting emulator with port number` *port-number*”.

   The application is now deployed and started on the connected device. You are ready to attach a debugger.

   See Attach a Command Line Debugger, Attach a Graphical Debugger and Sample CLDC Debugging Session

## Attach a Command Line Debugger

Start the application on the connected device as described in On-device Debugging Procedure. Attach the Java debugger (jdb) from the command line as follows:

```
jdb -connect com.sun.jdi.SocketAttach:port=51307
```

The above command assumes the host is 127.0.0.1 (localhost), and the debugging port is still set to the default of 51307. If you changed the device’s port number property, enter that value.

See Attach a Graphical Debugger and Sample CLDC Debugging Session.

## Attach a Graphical Debugger

This procedure describes how to attach the NetBeans graphical Java debugger (JPDA).

1. If you want to set line breakpoints, open the project you deployed on the connected device in NetBeans.

2. Select Debug > Attach Debugger.

3. In the Attach window, choose Java Debugger (JPDA) and click OK. You are prompted for connection settings
   - **Connector:** SocketAttach
   - **Host**: localhost, or leave as is
   - **Port**: the port number of the device running in debug mode.

   The debugger attaches to the device.

## Sample CLDC Debugging Session

This procedure provides instructions for running the FPDemo sample project on a device running Windows Mobile. You can use this procedure with any project.

1. In the Java ME Platform SDK, select File > Open Sample Project > FPDemo.
2. In the Projects window, right-click on FPDemo and select Properties.
3. Choose the Platform category. From the Device drop-down menu, select the name of the connected device. Click OK.
4. Select FPDemo and select Run > Run Project in Debug Mode, or click the Run Main Project in Debug mode icon on the toolbar.
5. Open a Windows shell.
6. Start `jdb` with the following command:

   ```
   db -sourcepath installdir\apps\FPDemo\src -connect
   com.sun.jdi.SocketAttach:hostname=127.0.0.1,port=port-number
   ```

   In this case the port number is 51307 to match the default debugging port number.
7. Set the breakpoint at the place where the demo handles its menu commands:

   ```
   stop in calculator.CalculatorMIDlet.commandAction
   ```
8. In the FPDemo application, enter two numbers, choose an operation, and invoke the Calc command.

   The debugger displays the place where the execution stops.
9. To step through the source past the point where the result is calculated, use the `next` command until you get past the following line:

   ```
   res = n1 <op> n2;
   ```
10. Check that the input and the calculated values are correct:

```
eval n1, eval n2, eval res
```

11. Override the calculated value as follows:

    `set res =` *new-value*

12. Clear the breakpoint as follows:

    ```
    clear calculator.CalculatorMIDlet.commandAction
    ```

13. Let the application continue:

    ```
    cont
    ```

    You can see that the application displays the overridden result.

14. Exit the debugger:

    ```
    exit
    ```

15. Exit FPDemo.

CHAPTER **17**

# Command Line Reference

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

- cbsreceive
- mmsreceive
- smssend
- cbssend
- mmssend

## Launching the SDK

The SDK can be launched from the command line as follows:

*installdir*`/toolbar/bin/Java_platform_ME_SDK.exe`

You can use the `--fontsize` argument to change the user interface fonts. For example:

*installdir*`/toolbar/bin/Java_platform_ME_SDK.exe --fontsize 18`

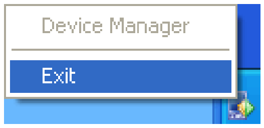
This setting does not affect the font size of the JavaHelp contents.

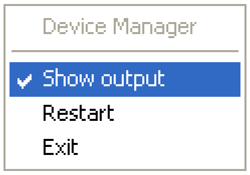
## Running the Device Manager

The device manager is a component that runs as a service. It detects devices (real or emulated) that conform to the Universal Emulator Interface Specification, version 1.0.2. The Device Manager automatically restarts every time your computer restarts.

You can manually launch *installdir*`/bin/device-manager.exe` from a script or the command line.

Right-click on the device manager icon in the system tray and you see the Show Output option. When it is selected the log is displayed in a console window.





## Managing Device Addresses (device-address)

*installdir*`/bin/device-address.exe` 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:

`device-address` *command arguments...*

Possible commands are as follows. Note, `ip` is currently the only supported address type.

| | |
|---|---|
| `add` *address_type address* | Add the specified address. |
| `del` *address_type address* | Delete the specified address. |
| `list` | List all address types. |
| `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`

## Running the Emulator From the Command Line

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

`emulator` *options*

The syntax for the `emulator` command is as follows:

The general options are as follows:

- `-classpath` *path*

  `-cp` *path*

  Specifies a search path for application classes. The path consists of directories, ZIP files, and JAR files separated by colons.

- -D*property=value*

  Sets a system property value.

- `-help`

  Display a list of valid options.

- `-version`

  Display version information about the emulator.

- `-Xdevice:`*skin-name*

  Run an application on the emulator using the given device instance name. See TABLE 8-1.

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

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

- `-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:\Java_ME_platform_SDK_3.0\apps\Games\dist\Games.jad
```

`emulator.exe` also supports MIDlet Options, CDC Options, and Debugging and Tracing Options.

## 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, JavaTest™ or Java Device Test Suite (JDTS). 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. Once the emulator starts, it queries the test harness, which then downloads and installs the TCK MIDletAgent.

- `-Xdescriptor:`*jad-file*

  Run an application locally using the given JAD file.

- `-Xdomain:`*domain-name*

  Set the MIDlet suite's security domain.

- `-Xjam:`*command=application*

  Run an application remotely using the Application Management Software (AMS) to run using OTA provisioning. If no application is specified with the argument, the graphical AMS is run. The commands are as follows:

  `install=`*jad-file-url* `| force | list | storageNames|`

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

  - `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.

  Also:

  `run=[`*storage-name* `|` *storage-number*`]`

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

  `remove=[`*storage-name* `|` *storage-number* `| all]`

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

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

  Install, run, and remove the application with the specified JAD file. Specifying `transient` causes the application to be installed and run and then removed three times.

This example includes OTA installation:

```
emulator.exe -Xjam:install=http://www.myserver.com/apps/MyApp.jad
-Xdevice:DefaultCldcMsaPhone2
```

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

See also Running the Emulator From the Command Line, CDC Options, and Debugging and Tracing Options.

## CDC Options

The following options apply to CDC projects.

- `-Xms`*n*

  Specifies the initial size *n*, in bytes, of the memory allocation pool, or heap size. This value must be a multiple of 1024 greater than 1 megabyte. Append the letter k or K to indicate Kilobytes or m or M to indicate megabytes. The default value is 64 megabytes (64M).

- `-Xmx`*n*

  Specifies the maximum size *n*, in bytes, of the memory allocation pool, or heap size. This value must be a multiple of 1024 greater than 1 megabyte. Append the letter k or K to indicate Kilobytes or m or M to indicate megabytes. The default value is 64 megabytes (64M).

- `-Xss`*n*

  Sets thread stack size to *n*.

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

  Run an Xlet application.

See also Running the Emulator From the Command Line and Debugging and Tracing Options.

## Debugging and Tracing Options

You can use the following options with the emulator for debugging and tracing both CLDC and CDC 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. This means that an application can be debugged from the very beginning.
- `-Xverbose:`*trace-options*

  Display trace output, as specified by a list of comma-separated options, as follows:
  - `gc` - Trace garbage collection
  - `class` - Trace class loading
  - `all` - Use all tracing options

This example shows debugging:

```
emulator.exe -Xdevice:DefaultCldcMsaPhone2 -Xdebug
-Xrunjdwp:transport=dt_socket, suspend=y,server=y,address=51307
-Xdescriptor:C:\Java_ME_platfrom_SDK_3.0\appps\Games\dist\Games.jad
```

See also MIDlet Options, CDC Options, and Running the Emulator From the Command Line.

## Building 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:

- Checking Prerequisites
- Compiling Class Files

- Preverifying Class Files

## Checking Prerequisites

Before building and running an application from the command line, verify that you have a version no earlier than 1.6 of the Java SE software development kit. Make sure the `jar` command is in your path. To find the version of the development kit, run the `jar` command and then run `java -version` at the command line.

## Compiling Class Files

Use the `javac` compiler from the Java SE development kit to compile Java source files. You can use the existing Java ME Platform 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\cldcapi10.jar;..\..\lib\midpapi20.jar
   -d tmpclasses
   src\*.java
```

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

## Preverifying Class Files

The next step is to preverify the class files. The `bin` directory of the Java ME Platform 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
  -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.

## 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 class and resource files. To create the JAR file, use the `jar` tool that comes with the Java SE software development kit. The syntax is as follows:

`jar cfm` *file manifest* `-C` *class-directory* `. -C` *resource-directory* `.`

The arguments are as follows:

- *file* - JAR file to create.
- *manifest* - Manifest file for the MIDlets.
- *class-directory* - Directory containing the application's classes.
- *resource-directory* - Directory containing the application's resources.

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

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

Create a JAD file containing the appropriate attributes as specified in the MIDP specification. You can use any text editor to create the JAD file. This file must have the extension `.jad`.

---

**Note –** You 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
```

## Command Line Security Features

The full spectrum of the Java ME Platform 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.

- Changing the Emulator's Default Protection Domain
- Signing MIDlet Suites (`jadtool.exe`)
- Managing Certificates (MEKeyTool)

## Changing the Emulator's Default Protection Domain

To adjust the emulator's default protection domain, use the following option with the `emulator` command:

`-Xdomain:`*domain-type*

Assigns a security domain to the MIDlet suite. Enter an appropriate security domain as described in Security Domains. For example, `-Xdomain:maximum`.

## Signing MIDlet Suites (`jadtool.exe`)

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

`jadtool.exe` 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.exe` 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.exe` *command*

The commands are as follows:

- `-help`

  Prints the usage instructions for `jadtool.exe`.

- `-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.

- `-addjarsig [-jarfile` *jarfile* `] [-keystore` *keystore* `] [-alias` *alias*`]`
  `[-storepass` *password*`] [-storetype` *PKCS11*`]`
  `[-keypass` *password*`] [encoding` *encoding*`]`
  `[-inputjad` *input-jadfile* `] [-outputjad` *output-jadfile*`]`

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

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

  Displays information about certificates in JAD and JPP files.

- `-addjppsig -keypass` *password* `-alias` *alias* `[-keystore` *keystore*`]`
  `[-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` - `%HOMEPATH%\.keystore`
- `-certnum` - `1`
- `-chainnum` - `1`

## Managing 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 keys can be used to facilitate secure HTTP communication over SSL (HTTPS).

Before using `MEKeyTool`, you must first have access to a Java Cryptography Extension keystore. You can create one using the Java SE `keytool` utility.

See http://java.sun.com/javase/6/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:

`mekeytool.exe -`*command*

The command keywords follow.

The *Java ME Platform SDK* contains a default ME keystore called `_main.ks,` which is located in the *installdir*`\runtimes\cldc-hi-javafx\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*`/javme-sdk/3.0/work/`*emulator-instance*`/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: `C:\Documents and Settings\`*user*`\.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 *user.home*`\.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.

## Generating 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 17-1 lists the `wscompile` options:

**TABLE 17-1** `wscompile` Options

| **Option** | **Description** |
|---|---|
| `-d` *output directory* | Specifies where to place generated output files |
| `-f:`*features* | Enables the given features |
| `-features:`*features* | Same as `-f:`*features* |

**TABLE 17-1** `wscompile` Options *(Continued)*

| Option | Description |
|---|---|
| `-g` | Generates debugging info |
| `-gen` | Same as `-gen:client` |
| `-gen:client` | Generates client artifacts (stubs, etc.) |
| `-httpproxy:`*host:port* | Specifies a HTTP proxy server (port defaults to 8080) |
| `-import` | Generates interfaces and value types only |
| `-model` *file* | Writes the internal model to the given file |
| `-O` | Optimizes generated code |
| `-s` *directory* | Specifies where to place generated source files |
| `-verbose` | Outputs messages about what the compiler is doing |
| `-version` | Prints version information |
| `-cldc1.0` | Sets the CLDC version to 1.0 (default). Float and double become String. |
| `-cldc1.1` | Sets the CLDC version to 1.1 (float and double are OK) |
| `-cldc1.0info` | Shows all CLDC 1.0 information and warning messages. |

**Note –** Exactly one `-gen` option must be specified. The `-f` option requires a comma-separated list of features.

TABLE 17-2 lists the features (delimited by commas) that can follow the `-f` option. The `wscompile` tool reads a WSDL file, compiled service endpoint interface (SEI), or model file as input. The Type of File column indicates which of these files can be used with a particular feature.

**TABLE 17-2** Command Supported Features (`-f`) for `wscompile`

| Option | Description | Type of File |
|---|---|---|
| `explicitcontext` | Turns on explicit service context mapping | WSDL |
| `nodatabinding` | Turns off data binding for literal encoding | WSDL |
| `noencodedtypes` | Turns off encoding type information | WSDL, SEI, model |
| `nomultirefs` | Turns off support for multiple references | WSDL, SEI, model |
| `novalidation` | Turns off full validation of imported WSDL documents | WSDL |
| `searchschema` | Searches schema aggressively for subtypes | WSDL |

**TABLE 17-2** Command Supported Features (`-f`) for `wscompile` *(Continued)*

| Option | Description | Type of File |
|---|---|---|
| `serializeinterfaces` | Turns on direct serialization of interface types | WSDL, SEI, model |
| `wsi` | Enables WSI-Basic Profile features (default) | WSDL |
| `resolveidref` | Resolves `xsd:IDREF` | WSDL |
| `nounwrap` | No unwrap. | WSDL |

### Examples

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

## Running the Payment Console From the Command Line

The Payment Console displays payment related transactions sent from a mobile application that implements Payment API (JSR 229) features. This console monitors Payment Update File requests and Premium Priced SMS payments. It is a command line version of the console discussed in Running the Payment Console.

The Device Manager must be running before you launch the Payment Console.

To launch the Payment Console from the command line, use *installdir*`/bin/payment-console.exe`.

## Virtual Machine Memory Profiler (Java Heap Memory Observe Tool)

The Java Heap Memory Observe Tool (also known as the Heap Walker) records detailed information about the Java heap at a specific point in the virtual machine execution. It collects and displays:

- global pointers
- data for all objects (classes, sizes, addresses and references)
- addresses for all roots

- names of all classes

Please note the following:

- The memory profiler uses the same connection technology as the debugger (see On-device Debugging Procedure). Because they use the same transport layer, the memory profiler and Java debugger cannot be used simultaneously.

---

**Note –** For the memory profiler, the `Xrunjdwp -suspend` option must be set to *n*.

---

- The memory monitor slows down your application startup because every object created is recorded.
- The memory usage you observe with the emulator is not exactly the same as memory usage on a real device. Remember, the emulator does not represent a real device, it is just one possible implementation of its supported APIs.

See also: Running the Java Heap Memory Observe Tool

## Running the Java Heap Memory Observe Tool

The Java Heap Memory Observe tool is launched from the command line as follows:

1. The following command starts the emulator from the \bin directory and opens a sample application:

   ```
   emulator.exe -Xdevice:DefaultCldcMsaPhone1 -Xdebug
   -Xrunjdwp:transport=dt_socket,suspend=n,server=y,address=51307
   -Xdescriptor:C:\Java_ME_platform_SDK_3.0\apps\Games\dist\
   Games.jad -Xdomain:maximum
   ```

2. In a separate command window, start the memory profiler from the \bin directory. The port number must match the address in the `Xrunjdwp` call:

   ```
   memory-profiler.exe -host 127.0.0.1 -port 51307
   ```

   The profiler window opens.

3. Click the Connect button and wait for the profiler to attach to the process. It may take several seconds. After the connection, press Resume.

4. To take a snapshot of the heap, click the Pause button on the bottom right. The virtual machine is suspended while the profiler collects the snapshot information. The memory panel is then repopulated and the Resume button becomes active.

See Viewing the Heap Snapshot

## Viewing the Heap Snapshot

The memory monitor elements are as follows:

**memory panel** - At the top of the window you see a grid of rectangles representing memory blocks. This is the memory panel. The key on the top right indicates the meaning of each graphical image. For example, blocks that are completely black are at 100% utilization. Clicking a single block opens a dialog showing all the objects in that block.

**loaded classes** - A list of loaded classes is displayed in the lower-left corner. Choosing a class from the list causes the location of all objects in the class to be displayed in class objects list immediately to the right.

**class objects** - The class objects list is populated when you select a class from the list of loaded classes. Select an object to see the class details. These include the address of the object, its type, and all references to and from the object. If the object is live, the "Show path from the root" button is enabled. Clicking this button opens the Path from the Root window, which displays dependencies that prevent this object from being garbage collected.

**statistics** - At the bottom right of the Memory Observer window, click the Statistics button to see a table showing the information for each class. Some objects are internal to the virtual machine. For each class, you see the Object number, size of all objects in the heap, the average size of the object, the percentage of the heap used by the selected class, the percentage of objects live in the selected class, and the percentage of objects that are in the old generation.

See Virtual Machine Memory Profiler (Java Heap Memory Observe Tool) and Running the Java Heap Memory Observe Tool

## Running WMA Tool

To send and receive SMS, CBS, and MMS messages from the command line, use *installdir*`/bin/wma-tool.exe`.

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

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

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

- 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 sending commands send the specified message and exit. The receiving commands print incoming messages until they are explicitly stopped.

Each command has its own arguments.

## smsreceive

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

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

`-t` *timeout*. Non-interactive mode, waits only *timeout* seconds for messages.

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

`-q` Quiet mode.

## cbsreceive

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

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

`-t` *timeout*. Non-interactive mode, waits only *timeout* seconds for messages.

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

`-q` Quiet mode.

## mmsreceive

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

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

`-t` *timeout*. Non-interactive mode, waits only *timeout* seconds for messages.

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

`-q` Quiet mode.

## smssend

`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.

## cbssend

cbssend *target_port* *message_content*

*target_port*

Port of the target phones to send the message to. 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.

## mmssend

`mmssend` *-application_id* *message_subject* *options*

*-application_id*

Application ID of the message. Mandatory first argument.

*-message_subject*

Subject of the message. Mandatory second argument.

**Options**

`-to` *target_phone*

"to" target phone number. Any number of these options can be used.

`-cc` *target_phone*

“cc” target phone number. Any number of these options can be used.

`-bcc` *target_phone*

“bcc” target phone number. Any number of these options can be used.

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

Appends text part to the message. Any number of these options can be used. Contains:

- *content ID*: content ID of this message part
- *encoding*: text encoding
- *text*: text of the message

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

Appends binary part to the message with content loaded from the given file. Any number of these options can be used. Contains:

- *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

CHAPTER 18

# Logs

Java ME Platform SDK uses the log4j logging facility to manage three types of logs:

- Java ME Platform SDK GUI Logs
- Device Manager Logs
- Device Instance Logs

## Java ME Platform SDK GUI Logs

The default location for the messages log is: `C:\Documents and Settings\`*USER_HOME*`\javame-sdk\toolbar\3.0\var\log`.

See also: Device Manager Logs and Device Instance Logs

## Device Manager Logs

The device manager log is placed into `toolkit-lib/process/device-manager/log`. Logging levels can be configured in the following XML file: `toolkit-lib/process/device-manager/conf/log4j.xml`. Priority value for category `com.sun` or VM can be set to 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>
```

See also: Java ME Platform SDK GUI Logs and Device Instance Logs

## Device Instance Logs

Each device (or emulator) instance writes its own log into its directory under *USER_HOME*`/javame-sdk/3.0/work/`*emulator-instance*. See `/javame-sdk/3.0/work`.

Like a device manager log it can be configured. Edit *installdir*`/toolkit-lib/modules/emulator-cldc/conf/log4j.xml` as described for Device Manager logs.

See also: Java ME Platform SDK GUI Logs and Device Manager Logs

CHAPTER 19

# JSR Support

The Java ME Platform 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.

See TABLE 19-1 for a full list of supported APIs. The 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.

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

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).

JSRs that are not part of the platform are referred to as “optional packages.“ All optional packages are supported onthe CLDC/MIDP Platform.

In Java ME SDK version 3.0 the CDC platform does not support JSR 239 (Java Binding for OpenGL ES API, and JSR 280 (XML API for Java ME).

See also TABLE 19-1.

# JCP APIs

**TABLE 19-1** 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` |
| JSR 177, SATSA | *Security and Trust Services API for Java ME* `http://jcp.org/en/jsr/detail?id=177` |
| JSR 179, Location | *Location API for Java ME* `http://jcp.org/en/jsr/detail?id=179` |
| JSR 180, SIP | *SIP API for Java ME* `http://jcp.org/en/jsr/detail?id=180` |
| JSR 184, 3D Graphics | *Mobile 3D Graphics API for J2ME* `http://jcp.org/en/jsr/detail?id=184` |
| JSR 185, JTWI 1.0 | *Java Technology for the Wireless Industry* `http://jcp.org/en/jsr/detail?id=185` |
| 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=218` |

**TABLE 19-1** Supported JCP APIs *(Continued)*

| JSR, API | Name, URL |
|---|---|
| JSR 218, CDC | *Connected Device Configuration* `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 229, Payment | *Payment API* `http://jcp.org/en/jsr/detail?id=229` |
| JSR 234, AMMS | *Advanced Multimedia Supplements* `http://jcp.org/en/jsr/detail?id=234` |
| JSR 238, MIA | *Mobile Internationalization API* `http://jcp.org/en/jsr/detail?id=238` |
| JSR 239 | *Java Binding for OpenGL ES API* `http://jcp.org/en/jsr/detail?id=239` |
| JSR 248, MSA 1.0 | *Mobile Service Architecture* `http://jcp.org/en/jsr/detail?id=248` |
| JSR 256 | *Mobile Sensor API* `http://jcp.org/en/jsr/detail?id=256` |
| JSR 280, XML API | *XML API for Java ME* `http://jcp.org/en/jsr/detail?id=280` |

CHAPTER **20**

# JSR 75: PDA Optional Packages

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

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

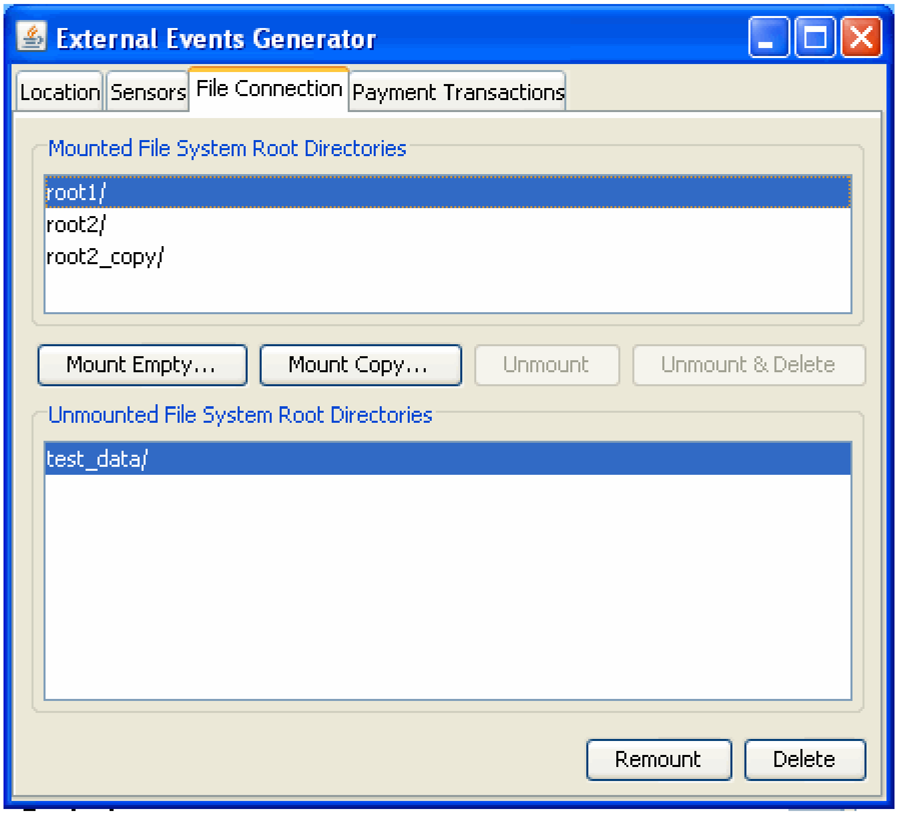
The files that can be accessed using FileConnection are stored in subdirectories of *USER_HOME*`/javame-sdk/`3.0`/work/`*emulator-instance*`/appdb/filesystem`. For example, the `DefaultCldcPhone1` emulator skin comes with a root directory installed called `root1`, which contains a `Readme` file and an empty directory named `photos`. The full path of the file is: *USER_HOME*`/javame-sdk/`3.0`/work/`*emulator-instance*`/appdb/filesystem\`root1\photos.

---

**Note –** If multiple instances of the same emulator skin run simultaneously, the Java ME Platform SDK generates unique file paths for each one. For instance, the first directory is named `DefaultCldcPhone1` and the second instance is named `DefaultCldcPhone2`.

---

Each subdirectory of `filesystem` is called a *root*. The Java ME Platform SDK provides a mechanism for managing roots. While the emulator is running, choose View > External Events Generator from the emulator window's menu. A utility window opens. Click the File Connection tab.



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. A dialog opens asking for the name of this root. 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.

## PIM API

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

*USER_HOME*`/javame-sdk/3.0/work/`*emulator-instance*`/appdb/filesystem/`*skin*`/pim.`

This directory is shared by all running emulators. Lists are stored in subdirectories of the `contacts`, `events`, and `todo` directories. For example, a contact list called Contacts is contained in *USER_HOME*`/javame-sdk/3.0/work/`*emulator-instance*`/appdb/filesystem/`*skin*`/Contacts`:

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

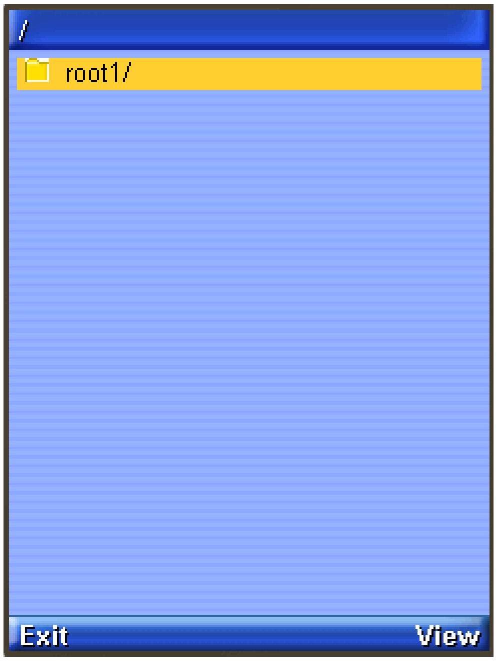
## Running PDAPDemo

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

## Browsing Files

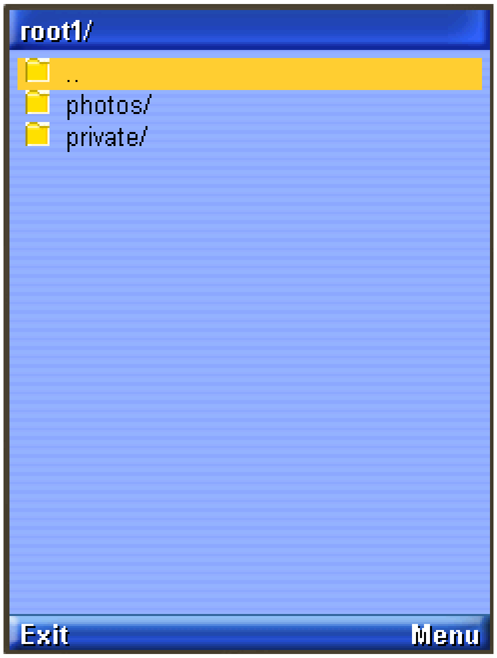
To run the file browser, you'll need to give the MIDlet appropriate security authorization, if you have not already done so. Right-click on your project, choose Properties, and select Specify the Security Domain. If necessary, select the maximum domain and press OK.

Now open and run the `PDAPDemo` project. Launch the `FileBrowser` MIDlet. You see a directory listing, and you can browse through the available directories and files. By default there is one directory, `root1`. This directory is located at *USER_HOME*`/javame-sdk/3.0/work/`*emulator-instance*`/appdb/filesystem/ro`ot1.



Select the directory and press the View soft button to enter it.

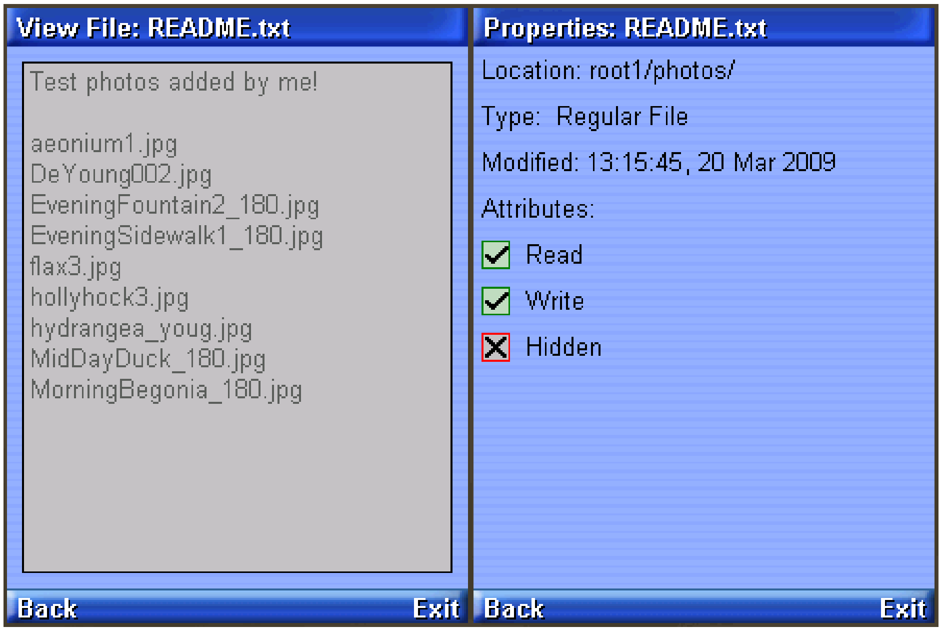
The directories `photos` and `private` are empty by default. You can add files and root directories and they will be visible to the JSR 75 File API. See Chapter 20 for more information. (This demo shows a README and some JPEGs that were added to the local `photos` directory.)



root1/

..
photos/
private/

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. Here we view `README.txt`:



View File: README.txt

```
Test photos added by me!

aeonium1.jpg
DeYoung002.jpg
EveningFountain2_180.jpg
EveningSidewalk1_180.jpg
flax3.jpg
hollyhock3.jpg
hydrangea_youg.jpg
MidDayDuck_180.jpg
MorningBegonia_180.jpg
```

Properties: README.txt

Location: root1/photos/

Type: Regular File

Modified: 13:15:45, 20 Mar 2009

Attributes:

✓ Read

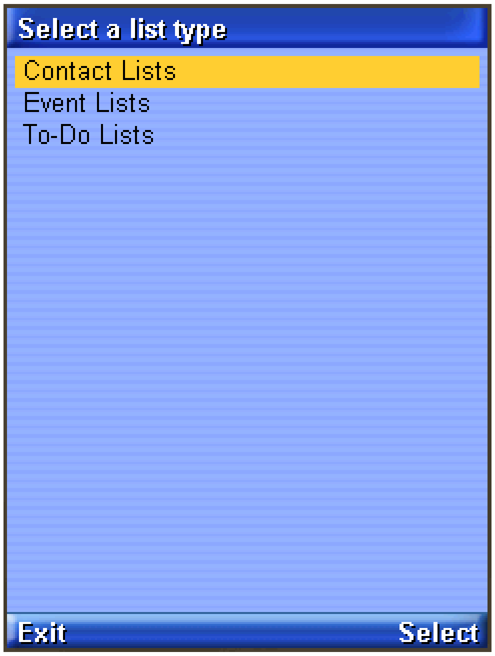
✓ Write

✗ Hidden

## The PIM API

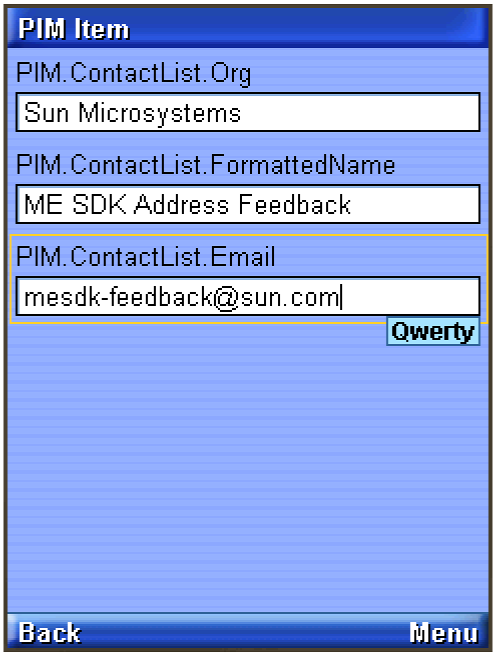
The JSR75 PIM APIs example demonstrates how to access personal information, like contact lists, calendars, and to-do lists. After you launch the example, choose a type of list from the main menu.

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



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

To add an item, choose New from the menu. The application prompts you for a Formatted Name for the item. You can add more data fields to this item using option 3, Add Field, in the menu. You see a list of field names. Pick as many as you like. You can fill in the field at any time.


PIM Item

PIM.ContactList.Org

Sun Microsystems

PIM.ContactList.FormattedName

ME SDK Address Feedback

PIM.ContactList.Email

mesdk-feedback@sun.com

To save the list item, choose Commit (option 5) 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 the *USER_HOME*/`javame-sdk/3.0/work/`*emulator-instance*`/appdb/filesystem/`*skin*`/pim` directory. See Chapter 20 for more information.

The PIM API allows for exporting contact, calender, and to-do items in a standard format. The exact format depends on the list type. When you are viewing an item in any list, the menu contains a command for viewing the exported item.

For example, when you are viewing a contact list item, the menu contains Show vCard. When you choose this command, the exported item is shown on the screen. Calendar items and to-do items both get exported as vCalendar.

CHAPTER 21

# JSR 82: Bluetooth and OBEX Support

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

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

## Bluetooth Simulation Environment

The Java ME Platform 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 Running the Bluetooth Demo.

## OBEX Over Infrared

The Java ME Platform SDK implements OBEX transfer over simulated Bluetooth and infrared connections. The simulated infrared connection follows the IrDA standard (see `http://www.irda.org/`). Simulated infrared transfers can take place between multiple running emulators.

## Setting Bluetooth and OBEX Preferences

The Java ME Platform SDK enables you to configure the Bluetooth and OBEX simulation environment. Choose Edit > Preferences and select Bluetooth/OBEX to display the following window.

### OBEX Preferences

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

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

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

### Bluetooth Internal Properties

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

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

## Bluetooth System Properties

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

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

## Bluetooth BCC Properties

The Bluetooth Control Center (BCC) controls Bluetooth settings. Some devices might provide a GUI to customize Bluetooth settings. In the Java ME Platform SDK, the BCC is configured using the BCC Properties tab of the Bluetooth preferences. The properties are as follows.

**TABLE 21-1** BCC Properties

| Property | Description |
|---|---|
| Enable Bluetooth support | If this property is disabled, `LocalDevice.getLocalDevice()` throws a `BluetoothStateException` and no connections can be created. This is useful to test the behavior of your application on devices that support JSR 82 but might have the Bluetooth feature turned off. |
| Device is discoverable | Indicates whether or not this emulator can be discovered by other emulators. |
| Friendly name | A human-readable name for the emulator in the simulated Bluetooth environment. If the name is left blank, the emulator does not support the friendly name feature. |
| Encryption | Determines whether connection encryption is supported (`on`) or not (`off`). In addition, the `force` settings means all connections must be encrypted. See the documentation for `RemoteDevice`’s `encrypt()` method for details. |
| Authorization | Similar to the Encryption property. See `RemoteDevice`’s `authorize()` method. |
| Authentication | Similar to Encryption and Authorization. See `RemoteDevice`’s `authenticate()` method. |

## 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.

For example, use DefaultCldcMsaPhone1 to launch Bluetooth Demo, then choose Server. The emulator asks you if you want to allow a Bluetooth connection. Choose Yes. The server starts and displays a list of images. At the beginning, none of the images are available on the Bluetooth network

Select the images you want to make available.

From the menu choose Publish image (or type or click 1). The icon color changes from purple to green, indicating it is published.

Use DefaultCldcMsaPhone2 to launch Bluetooth Demo, then select Client. The MIDlet tells you it's ready to search for images. Click the Find soft button. The MIDlet finds the other emulator and get a list of images from it. Select one from the list and choose Load.

- 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.

CHAPTER **22**

# 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 with respect to 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://jcp.org/en/jsr/detail?id=135`

## Media Types

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

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

## Adaptive Multi-Rate (AMR) Content

The Java ME Platform SDK simulates support for Adaptive Multi-Rate (AMR) content (`http://www.ietf.org/rfc/rfc3267.txt`). Although the Java ME Platform SDK cannot decode AMR content, the implementation returns a player for AMR content when requested.

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

## Media Capture

The Java ME Platform SDK emulator supports audio and video capture. Audio capture is supported by using the capture capabilities of the system upon which the emulator runs.

Video capture is supported by simulating a camera input.

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

## 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 Java ME Platform 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 in the `MobileMediaAPI` demonstration application.

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

## Ring Tones

MMAPI can be used to play ring tones, as demonstrated in Simple Tones and Simple Player. The ring tone formats mentioned here are in common use. You can download ring tones or create your own.

### Download Ring Tones

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

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

### Ring Tone Formats

This section provides samples of several formats

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

  `http://www.convertyourtone.com/rtttl.html`
- Nokia Composer

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

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

- Ericsson Composer

  Beethoven's 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 sympony:

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

## Running the MMAPI Sample Project

The following projects demonstrate ways to implement MMAPI 1.1. See Running AudioDemo, Running MMAPIDemos.

### 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.

## Running MMAPIDemos

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

- Simple Tones
- Simple Player
- Video
- Pausing Audio Test

### 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.

### Simple Player

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

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

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

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

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

The Simple Player includes a common set of commands that control media playback. The commands are available from the Simple Player menu, and some have associated keypad buttons. TABLE 22-1 describes these commands.

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

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

**TABLE 22-1** Simple Player Commands

| Command | Menu Item | Description |
|---|---|---|
| Mute/Unmute | 1 | Turns off sound but the file continues to play. This command toggles to Unmute. |
| Play | 2 | |
| Volume | 3 | Increases or decreases loudness. |
| META Data | 4 | Displays information provided by the media file such as copyright information, title, and track list. |
| Stop in 5 seconds | 5 | Pauses the audio play in five seconds when set during playback. |
| Loopmode | 6 | |
| Rate | 7 | Alters the rate of speed of playback. |
| Tempo | 8 | Increases or decreases the tempo of the tone sequence or MIDI file. |
| Pitch | 9 | Lowers or raises the notes in a MIDI file. |
| Skip Forward | | Skips forward five percent of the duration of the media file. The sound track syncs to the video |
| Skip Backward | | Skips backward five percent of the duration of the media file. The sound track syncs to the video. |
| Rewind | | Returns to the start time of the audio playback. |
| Quick Help | | Displays a list of commands and keypad buttons. |

## Video

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

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

- **Animated GIF** - Canvas [jar] shows an animated GIF in a Canvas. A simple indicator shows the progress through the animation. Choose Snapshot to get a still image of the current appearance. The snapshot is shown briefly, then the display goes back to the animation.
- **Video Capture** - Form simulates capturing video from a camera or other source and showing it as an Item in a Form. Choose the Snapshot command to take a snapshot of the captured video. The snapshot will be placed in the form following the video capture.
- **Video Capture** - Canvas simulates capturing video from a camera or other source and showing it in a Canvas. Choose Snapshot to get a still image of the current appearance. The snapshot is shown briefly, then the display goes back to the video capture.
- **MPEG1 Video** - Form, MPEG1 Video - Canvas

  The MPEG1 applications obtain MPEGs from the web, so if you are behind a firewall, you must configure the emulator's proxy server settings, as described in Configuring the Web Browser and Proxy Settings.

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

## Attributes for MobileMediaAPI

The `MobileMediaAPI` applications have the following editable attributes. Right-click on the project and select Properties. Select Application Descriptor and view the Attributes tab.

**TABLE 22-2** Descriptions of MMAPI-specific MIDlet attributes

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

**TABLE 22-2** Descriptions of MMAPI-specific MIDlet attributes *(Continued)*

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

## Running the Multimedia Sample Project

This MIDlet simply plays different multimedia samples. The sample formats are: AMR narrow band, MIDI audio, Wave audio, MPEG video, and animated GIF.

CHAPTER 23

# JSR 172: Web Services Support

The Java ME Platform 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.

## Generating Stub Files from WSDL Descriptors

The Java ME Platform SDK 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. The following is a general procedure for adding stubs:

1. In the Projects window, expand the tree for a project.
2. Right-click on the Source Packages node and select New > Other.
3. In the Categories pane select Other, and in the File Types area choose Mobile Webservice Client.
4. In the Generate J2ME Webservice Stub page, you can either:
   - Click Running Web Service and enter the URL for the WSDL
   - 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, then click Finish. The new package appears in the project and includes an interface file and a stub file.
5. You can now edit your source files to call the content the stub provides, then build and run.

See Creating a New Mobile Web Service Client for a step by step process, or see Running JSR172Demo and view the demo source files.

## 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.

1. Select File > New Project, choose MIDP application, and click Next. Name your project and ensure Create Hello MIDlet is checked. Click Finish.
2. Right-click on the new project's Source Packages node and select New > Other.
3. In the Categories pane select Other, and in the File Types area choose Mobile Webservice Client.
4. In the Generate J2ME Webservice Stub page:
   - Click Running Web Service and in the WSDL URL field, enter:

     `http://www.xmlme.com/WSShakespeare.asmx?WSDL`
   - In the Package field, enter `testws`. This is the package name.

   Click Finish. The new package appears in Source Packages and includes `Shakespeare.java` and `Shakespeare_Stub.java`.
5. Edit `HelloMIDlet.java` as follows:
   - At the beginning, add the following import declaration:

     `import testws.*`
   - 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);
```

```
t.addCommand(exitCommand);
t.setCommandListener(this);
display.setCurrent(t);
```

6. 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.

## Running JSR172Demo

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

If you are using a proxy server, you must configure the emulator's proxy server settings as described in Configuring the Web Browser and Proxy Settings. Build and run the example.

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

CHAPTER **24**

# 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 Java ME Platform SDK emulator fully supports SATSA. This topic describes how you can use the Java ME Platform 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://java.sun.com/j2me/docs/satsa-dg/`.

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

## Card Slots in the Emulator

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

The Java ME Platform 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 Java ME Platform 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 ports are 9025 for slot 0 and 9026 for slot 1. These port defaults are a property of the device. To alter the defaults, go to the device directory:

`C:\Documents and Settings\`*user*`\javame-sdk\` *3.0*`\`*directory-number*

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

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

## Java Card Platform Simulator (`cref`)

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

*installdir*`\bin\cref.exe`.

Hereafter we refer to it as simply cref. The basic procedure for testing SATSA applications with the Java ME Platform 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`.

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

## 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*`\` *emulator-instance*`\appdb` directory. For example, an access control file for slot 0 might be:

```
C:\Documents and Settings\User\javame-sdk\3.0\work\emulator_instance\
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.

### 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
   }
```

## 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 signatiure
                   ...
                 }
         }
         ...
         pin_apdu {
                 id number
            verify | change | disable | enable | unblock
            four hexadecimal numbers
                 ...
         }
         ...
         pin_jcrmi {
                id number
            verify | change | disable | enable | unblock
            method name and signature
                ...
           }
        ...
        }
    ...
}
```

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

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

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

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

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

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

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

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

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

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

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

## 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
     }
   }
}
```

## 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 Java Card Platform Simulator (`cref`).

The following sections contain instructions for each menu choice for this demo. For each demo, follow this sequence:

1. Start the instance(s) of `cref` from the command line.
2. Be sure to set the project security domain to maximum.

   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. Run the project.

## APDUMIDlet

This MIDlet demonstrates communication with a smart card using Application Protocol Data Units (APDUs), small packets of data. `APDUMIDlet` expects to find two simulated smart cards. You can run the smart card simulator using `cref`, which is part of the Java Card Development Kit.

The `Mohair` application includes pre-built memory images that you can use with `cref`. The memory images contain Java Card applications with which `Mohair` interacts. The memory images are in the root directory of the `Mohair` project.

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 bin\cref -p 9025 -i apps\SATSADemos\demo2.eeprom
   start bin\cref -p 9026 -i apps\SATSADemos\demo2.eeprom
   ```

   Note that the port number arguments (9025 and 9026 in this example) must match the SATSA port numbers. Also, make sure you use the correct path to `demo2.eeprom`.
3. Once 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.

## SATMIDlet

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

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

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` as follows:

   ```
   start bin\cref -i apps\SATSADemos\sat.eeprom
   ```

   Note that the port number arguments (9025 and 9026 in this example) must match the SATSA port numbers. Also, make sure you use the correct path to `sat.eeprom`.

3. Once you have the smart card simulator running, run SATSADemos. Select SATMIDlet, select the Menu soft key and select Launch (1). Press Go when prompted.

## 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.

## MohairMIDlet

`MohairMIDlet` has two functions. The first, "Find slots", displays all the available card slots. Each slot has a number followed by 'C' or 'H' indicating whether the slot is cold-swappable or hot-swappable. After viewing the slots select Back to return to the first screen.

The second part of `MohairMIDlet`, SATSA-PKI Sign test, uses a smart card to generate a digital signature. As with the earlier demonstrations, you need to run `cref` with the right memory image to prepare for the connection from `MohairMIDlet`.

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. Type the following from the SDK installation directory:

   start `bin\cref -i apps\SATSADemos\sat.eeprom`
3. In the emulator, select SATSA-PKI Sign test. The following confirmation message appears:

   ```
   This certificate will be used: MohairAuth
   ```

   Select the OK soft key.
4. For PIN 1, type: `1234`

   Select the OK soft key. 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: MohairAuth
   ```

   Select the OK soft key.
6. For non repudiation key 1 PIN, type: `2345`

Select the soft menu and choose OK (option 2).

## 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 need to start up `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 bin\cref -p 9025 -i apps\SATSADemos\demo2.eeprom
   ```

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

CHAPTER 25

# JSR 179: Location API Support

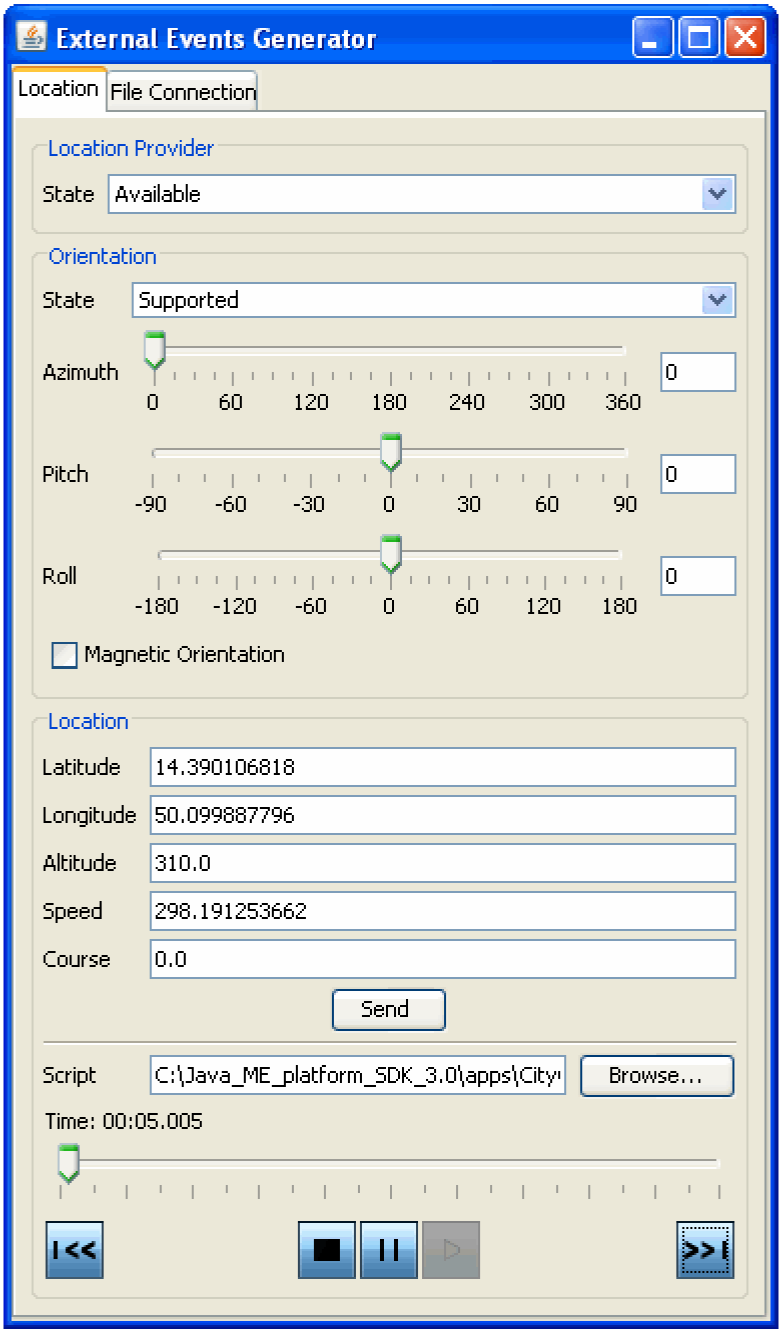
The JSR 179 Location API gives applications the opportunity to use a device's location capabilities. For example, some devices include Global Positioning System (GPS) hardware. Other devices might be able to receive location information from the wireless network. The Location API provides a standard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

## Setting the Emulator's Location at Runtime

You can specify the simulated location of the emulator while it is running. To do this, choose View > External Events Generator from the emulator window's menu. Click the Location tab.


External Events Generator
Location
File Connection
Location Provider
State
Available
Orientation
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
Magnetic Orientation
Location
Latitude
14.390106818
Longitude
50.099887796
Altitude
310.0
Speed
298.191253662
Course
0.0
Send
Script
C:\Java_ME_platform_SDK_3.0\apps\City
Browse...
Time: 00:05.005

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 Java ME Platform 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>
```

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

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

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

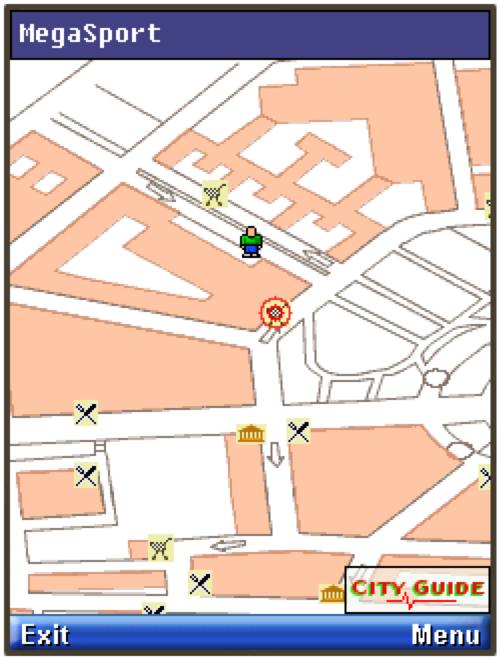
To test how your application handles unexpected conditions, try changing the State field in the Location Provider box to Temporarily Unavailable or Out of Service. When your application attempts to retrieve the emulator's location, an exception is thrown and you can see how your application responds.

## 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. In this demo we get the walker's location from an XML script named `citywalk.xml` (the event file) that submits the device location information.

Because location prompts occur frequently, it is best to run this demonstration in manufacturer (trusted) mode, as explained in 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.

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



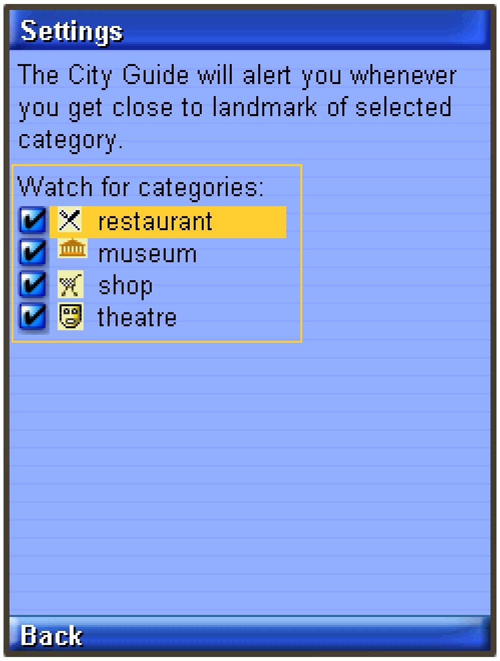
Choose View > External Events Generator from the emulator window menu. On the Location tab click the browse button. Select the following event file in *installdir*`\apps\CityGuide\citywalk.xml`.

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

The display shows four types of landmarks: restaurants, museums, shops, and theaters. To adjust the landmark display, open the soft menu and choose the Settings command.

Use the navigation keys to highlight a category, then use Select to check or uncheck an item.

When you are near a landmark (shown highlighted on the map), open the soft menu and choose the Detail command to see more information. See Chapter 25 for more details on location scripts.



Settings

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

Watch for categories:

- ✓ restaurant
- ✓ museum
- ✓ shop
- ✓ theatre

Back

CHAPTER 26

# JSRs 184, 226, and 239: Graphics Capabilities

The Java ME Platform SDK offers three APIs that provide comprehensive capabilities for interactive 2D and 3D graphics:

The Mobile 3D Graphics (JSR 184) API for J2ME, JSR 184, provides 3D graphics capabilities with a low-level API and a high-level scene graph API. This chapter provides a brief overview and general guidelines for working with JSR 184.

The Scalable 2D Vector Graphics (JSR 226) for J2ME, JSR 226, supports rendering sophisticated and interactive 2D content.

Java Bindings for OpenGL ES (JSR 239), JSR 239, provides a Java language interface to the open standard OpenGL ES graphics API.

# Mobile 3D Graphics (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`.

## 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.

### 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.

## 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.

## 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.

## 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. Superscape (`http://superscape.com/`) is one such vendor.

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

## Running Demo3D Samples

Demo3D contains three MIDlets that demonstrate JSR 184 features.

### 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 26-1 provide control over the game.

**TABLE 26-1** 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. |

See the source code for this example is at *installdir*`\apps\Demo3D\src\com\superscape\m3g\wtksamples\life3d\Life3D.java`.

### retainedmode

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

### 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. You might need to hold down the key to see an effect.

## Java Bindings for OpenGL ES (JSR 239)

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

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

## Scalable 2D Vector Graphics (JSR 226)

The Java ME Platform SDK emulator supports JSR 226, the Scalable 2D Vector Graphics API for J2ME. 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 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:

- **Ikivo Animator -** `http://www.ikivo.com/animator/`
- **Adobe Illustrator -** `http://www.adobe.com/products/illustrator/main.html`

## Running SVGDemo

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

### 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 at:

*device*`\appdb\filesystem\root1`

For your device location, see “`/javame-sdk/3.0/work`” on page 80. Launch the demo. The application displays the contents of `root1`. Select your SVG file and choose the Open soft key.

### Render SVG Image

Render SVG Image loads an SVG image from a file and renders it. Looking at the demo code you can see that the image is sized on the fly to exactly fit the display area. The output is clear and sharp.

### Play SVG Animation

This application plays an SVG animation depicting a Halloween greeting card. Press 8 to play, 5 to start, and 0 to stop. If you press 8, pressing 5 resumes the animation. If you press 0, pressing 5 starts the animation from the beginning.

The SVG file contains a description of how the various image elements evolve over time to provide this short animation.

In the following code sample, the JSR 226 `javax.microedition.m2g.SVGImage` class is used to load the SVG resource. Then, the `javax.microedition.m2g.SVGAnimator` class can take all the complexity of SVG animations and provides a `java.awt.Component` or `javax.swing.JComponent` which plays the animation. The `SVGAnimator` class provides methods to play, pause and stop the animation.

```
import javax.microedition.m2g.ScalableGraphics;
import javax.microedition.m2g.SVGImage;

...
String svgURI = ...;
SVGImage svgImage = (SVGImage) SVGImage.createImage(svgURI, null);
SVGAnimator svgAnimator = SVGAnimator.createAnimator(svgImage);

// If running a JSE applet, the target component is a JComponent.
JComponent svgAnimationComponent = (JComponent)
svgAnimator.getTargetComponent();
...

svgAnimator.play();
...
svgAnimator.pause();
...
svgAnimator.stop();
```

## 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.

## 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.

## Optimized Menu

In this demo, selected icons have a yellow border. As you move to a new icon, it becomes selected and the previous icon flips to the unselected state. If you navigate off the icon grid, selection loops around. That is, if the last icon in a row is selected, moving right selects the first icon in the same row.

This demo illustrates the flexibility that combining UI markup and Java offers: a rich set of functionality (graphics, animations, high-end 2D rendering) and flexibility in graphic manipulation, pre-rendering or playing.

In this example, a graphic artist delivered an SVG animation defining the transition state for the menu icons, from the unselected state to the selected state. The program renders each icon's animation sequence separately into off-screen buffers (for faster rendering later on), using the JSR 226 API.

With buffering, the MIDlet is able to adapt to the device display resolution (because the graphics are defined in SVG format) and still retain the speed of bitmap rendering. In addition, the MIDlet is still leveraging the SVG animation capabilities.

The task of defining the look of the menu items and their animation effect (the job of the graphic artist and designer) is cleanly separated from the task of displaying the menu and starting actions based on menu selection (the job of the developer). The two can vary independently as long as both the artist and the developer observe the SVG document structure conventions.

## Picture Decorator

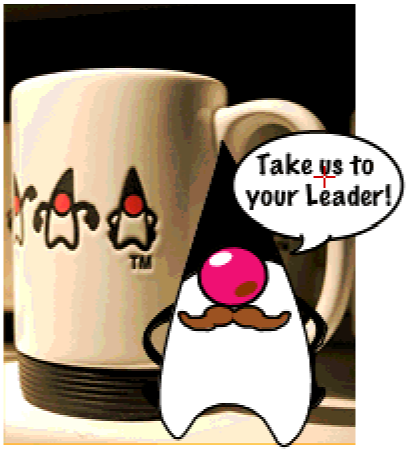
In this sample you use the phone keys to add decorations to a photograph. The key values are:

| | |
|---|---|
| 1 | key shrink |
| 2 | key next picture |
| 3 | key grow |
| 4 | key help |
| 5 | key horizontal flip |
| 6 | ley vertical flip |
| 7 | key rotate counter-clockwise |
| 8 | key previous picture |
| 9 | key rotate clockwise |
| # | display picker options |

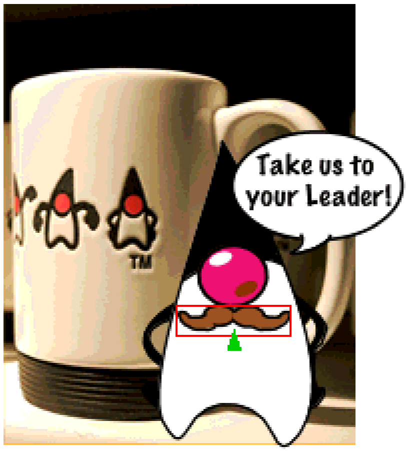
This demo provides 16 pictures for you to decorate.

Use the 2 and 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. This means it is selected and can be moved, resized, and manipulated.



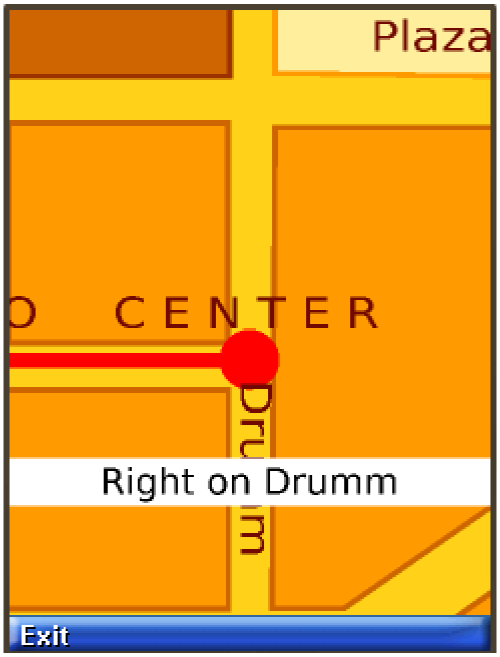
Use the navigation arrows to move the graphic. Use 1 to shrink the graphic, and 3 to enlarge the graphic. Use 5 or 6 to flip, and 7 or 9 to rotate. When you are satisfied with the position, press Select. Note that a green triangle appears. This is a cursor. Use the navigation keys to move the green triangle around the picture. When the cursor is over an object it is highlighted with a red box. Press Select. The red + indicates the object is selected.



To remove a decoration (a property), select an object, then click the Menu soft key. Press 2 to remove a property.

## 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.



## PlaySVGImageDemo

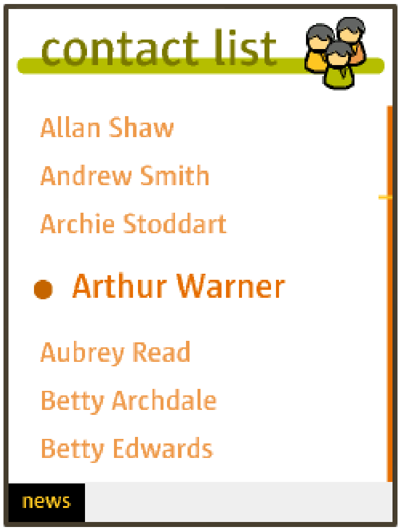
Displays a Duke image. Select the Virtual soft key to display a virtual navigation pad.

## Running SVGContactList

This application uses different skins to display the same contact list information and a news banner. The skins have different colors and fonts.

Select SVGContactlist(skin 1) or SVGContactlist(skin 2), then click Launch.

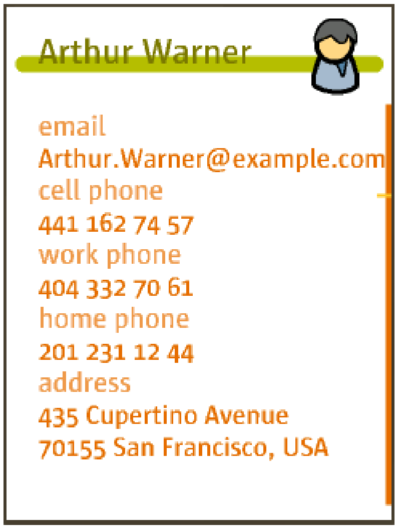
Use the up and down arrows to navigate the list of contacts. The highlighted name is marked with a special character (a > or a dot) and is displayed in a larger font.


contact list

Allan Shaw
Andrew Smith
Archie Stoddart
● Arthur Warner
Aubrey Read
Betty Archdale
Betty Edwards

news

Press the select button to see more information for the highlighted name.


Arthur Warner

email
Arthur.Warner@example.com
cell phone
441 162 74 57
work phone
404 332 70 61
home phone
201 231 12 44
address
435 Cupertino Avenue
70155 San Francisco, USA

Press select again to return to the contact list.

CHAPTER 27

# JSR 205: Wireless Messaging API (WMA) Support

The Java ME Platform 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 support for MMS messages as well.

This chapter describes the tools you can use to develop WMA applications. It begins by showing how to configure the emulator's support of WMA. Next, it describes the WMA console, a tool for testing WMA applications.

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

## 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 JSRs 120 or 205. You can, for example, use the WMA console to send SMS messages to a MIDlet running on the emulator.

See Launching the WMA Console and WMA Console Interface

### Launching the WMA Console

To launch the WMA console, select Tools > WMA Console. If the WMA Output window is not visible, select Window > Output > WMA Console Output.

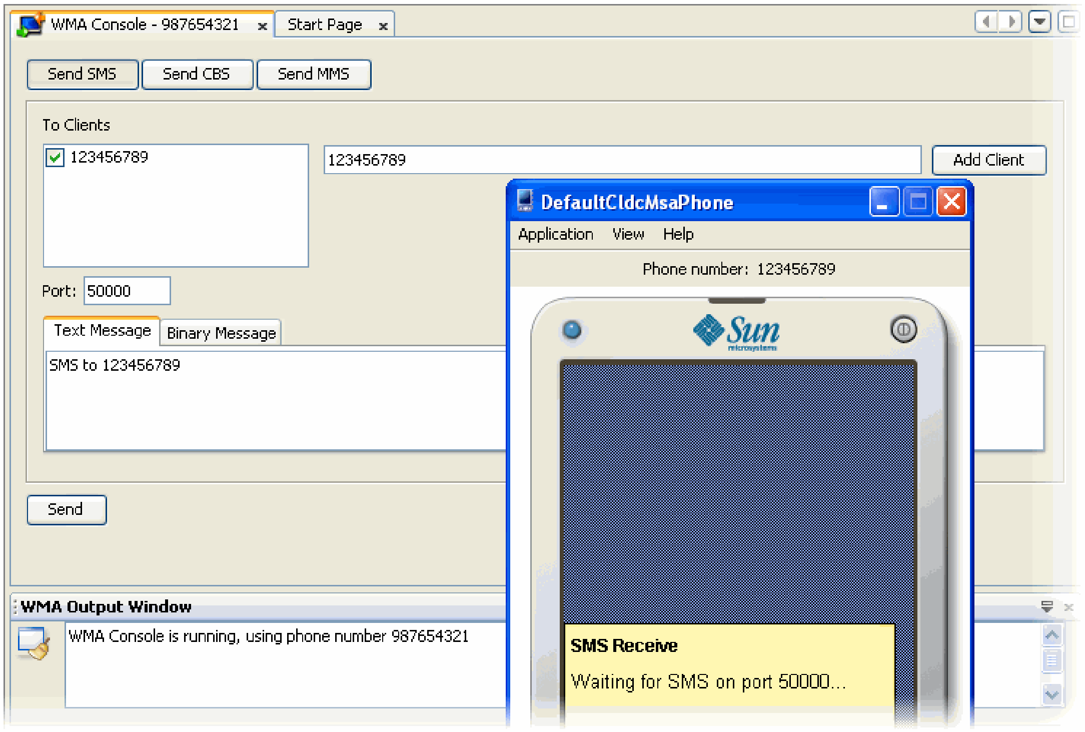
Note, WMA console operations can also be performed from the command line. See Running WMA Tool.

## WMA Console Interface

To view this interface, select Tools > WMA Console. If the WMA Output window is not visible, select Window > Output > WMA Console Output.

The WMA Console user interface has a tab for sending messages and an output window that displays incoming messages.

As shown here, the console has a phone number, and it is displayed as part of the WMA console tab label.



To Clients: ✓ 123456789

123456789

Port: 50000

SMS to 123456789

WMA Console is running, using phone number 987654321

Phone number: 123456789

SMS Receive

Waiting for SMS on port 50000...

To set the phone number, select Tools > Options > Miscellaneous. On the WMA Console tab, edit the Assigned Phone Number field and click OK. 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 SDK.

## Emulator Phone Numbers

Each running 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. By default, the first emulator instance has a phone number of 123456789. Subsequent instances of the emulator have unique numbers in ascending order.

## Sending a Text SMS Message

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

The window automatically lists the phone numbers of all running emulator instances. Select one or more destinations and enter a port number if you wish. Type your message and click Send.

## Sending a Binary SMS Message

To send the contents of a file as a binary message, click Send SMS to bring up the send window. Click the Binary SMS tab.

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

## Sending Text or Binary CBS Messages

Sending CBS messages is similar to sending SMS messages except that you don't need to choose recipients. To send a text or binary CBS message, click Send CBS in the WMA console. Specify a message identifier and enter the content of your message.

## 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 two tabs, one for recipients and one for content. On the Header tab, begin by filling in a subject and recipient.

To add more recipients, click the Add button. For example, to send a message to a running emulator whose number is 5550001, type 5550001 in the To field.

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

To add optional media files (Parts) to the message, click the Parts tab and click Add. Most media files will 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.

To remove a part, select it and press Remove.

## Receiving Messages in the 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.

# Running WMADemo

The WMADemo sample project shows how to send and receive SMS, CBS, and MMS messages.

The Java ME Platform SDK offers a flexible emulation environment to support messaging. Messages can be exchanged between emulator instances and can be generated or received using the WMA console utility.

Because this sample makes use of the push registry, you can't see all of its features just by using the Run button. Use the Run via OTA feature to install the application into the emulator in a process that mirrors how applications are installed on real devices.

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

1. To launch the console choose Tools > WMA Console.

2. Click on the Send SMS button in the WMA console window. Choose the number that corresponds to the emulator, probably 123456789. If you're not sure what number the emulator is using, look for a number above the emulator screen. Choose the number in the SMS message window, then fill in a port number of 50000. Type your text message in the Message field and click on Send.

3. The emulator asks if it can launch the `WMADemo` application. 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.

You can also use the WMA console to send and receive CBS and MMS messages.

If you are attempting to send text messages to `WMADemo` using the WMA console specify the following port numbers:

- For SMS specify the port number 50000.
- Use port 50001 for CBS messages.
- For MMS messages, use `example.mms.MMSDemo` as the application ID. This information is part of 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 example, to send an MMS message from the WMA console to the emulator, make sure that `WMADemo` has been installed using Run via OTA.

1. Launch the demo and choose MMS Receive.

2. In the WMA console, click on Send MMS to open the MMS composition window. Fill in a message subject, the application ID `example.mms.MMSDemo`, and the telephone number of the running emulator.

3. Click on the Parts tab. The WMA console allows you to select files from your hard disk to send as parts of the MMS message. Click Add to add a file to the message. Use the file browser to find the file you want to send and click OK.

4. Click on Send to send the message.

   The image and its information are displayed.

CHAPTER 28

# JSR 211: Content Handler API (CHAPI)

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

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

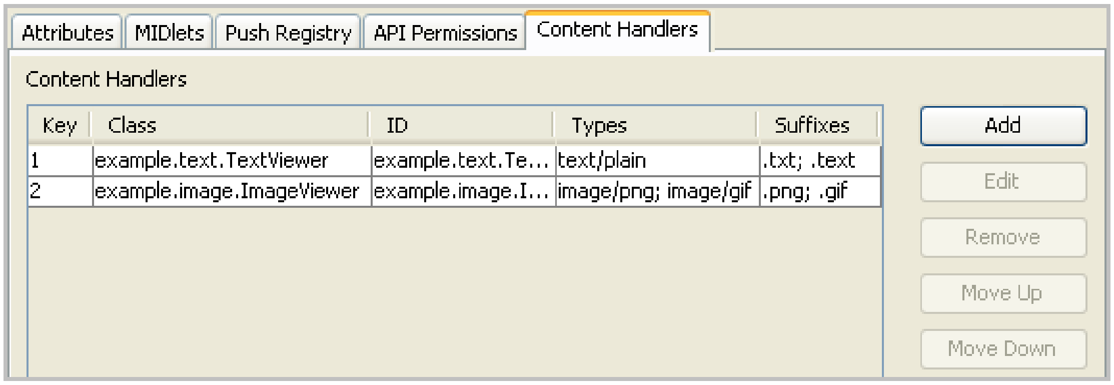
See Using Content Handlers and Running the CHAPIDemo Content Browser.

## Using Content Handlers

Follow these steps to work with content handlers.

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

   As shown below, two content handlers have been configured, one for TextViewer and one for ImageViewer.



Content Handlers

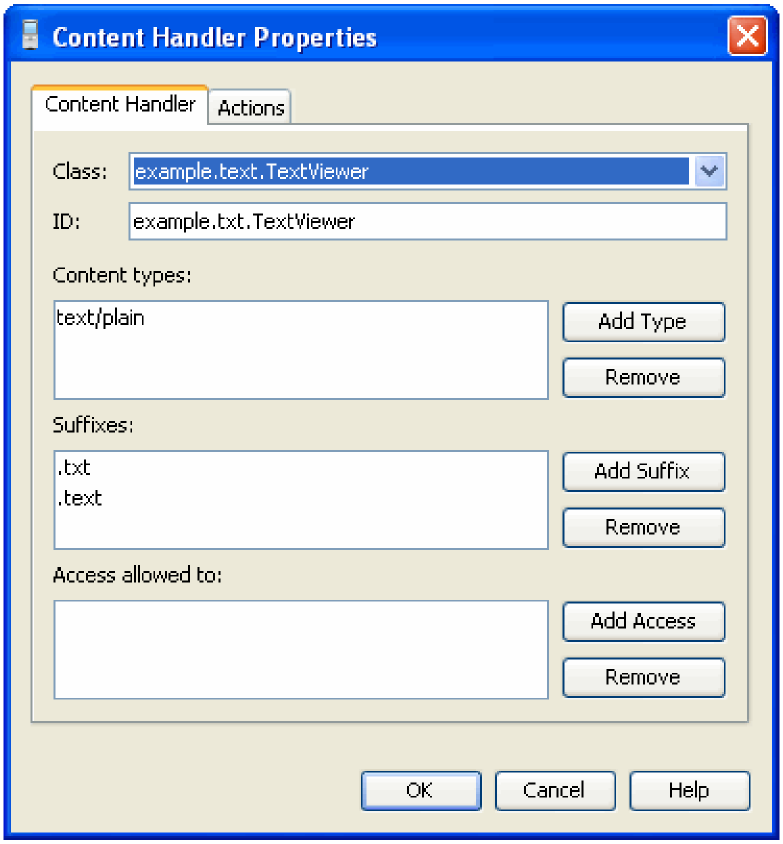
| Key | Class | ID | Types | Suffixes |
|---|---|---|---|---|
| 1 | example.text.TextViewer | example.text.Te... | text/plain | .txt; .text |
| 2 | example.image.ImageViewer | example.image.I... | image/png; image/gif | .png; .gif |

- 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 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 also Defining Content Handler Properties, Defining Content Handler Actions, and Running the CHAPIDemo Content Browser.

## 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, as shown here.


Class: example.text.TextViewer

ID: example.txt.TextViewer

Content types:

text/plain

Suffixes:

.txt
.text

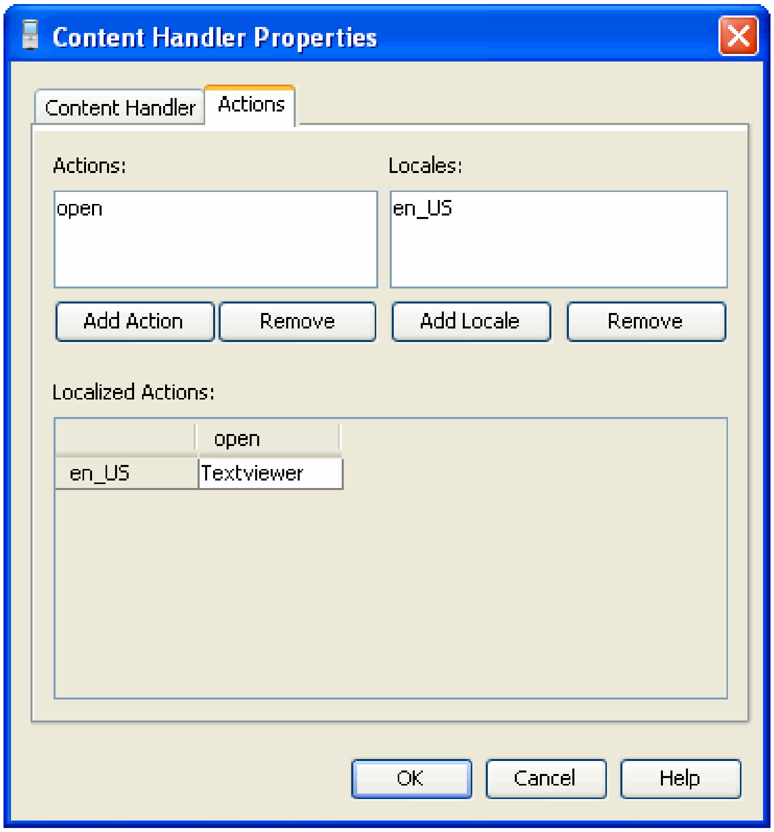
Access allowed to:

- In the Class field, choose a class name from the dropdown menu.
- ID is an identification string that can be used to 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.

## 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 on 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 will be provided. Localized actions is a grid which contains the human-readable action names for various locales. Each locale is represented by a row, while the actions are listed as columns. You can see all the human-readable action names for a particular locale by reading across a single row.

## Running the CHAPIDemo Content Browser

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

- In the user interface, select File > Open Sample Project > CHAPIDemo.

  In the device selector, right-click on a device, select Run Project OTA, and choose CHAPIDemo.

  - You are asked to enter the password for the keystore. Type: `keystorepwd`
  - You are asked to enter the password for the "dummyca" key pair alias within the keystore. Type: `keypwd`
  - On the Favorite Links page, choose CHAPI Demo. Press the menu soft button and choose 1, Go.

  The Text Viewer displays a Media Player URL and links to various media files.

- Select `Duke.png`. Use the arrows to highlight the link, then select the file. Using CHAPI, the `ImageViewer` MIDlet runs and displays the Duke image. Select the Back soft key to return to the Text Viewer.
- Install the Media Player to view media.
  - Select the URL http:handlers/MediaHandler.jad.

    Select the Menu soft button and select item 1, Go.
  - The application asks, "Are you sure you want to install Media Handler?" Select the Install soft key. For the rest of this demo, click Install if you are asked for confirmation.

    The installation finishes and you return to the Text Viewer.
- View different media files.

  Select a URL from the list, select the Menu soft button and select item 1, Go.

CHAPTER **29**

# JSR 238: Mobile Internationalization API (MIA)

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

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

Resources are stored in files in a format defined in JSR 238. The resource files are bundled as part of the MIDlet suite JAR file. The Java ME Platform SDK provides a resource manager that simplifies the job of creating and maintaining resource files.

## Setting the Emulator's Locale

You can change an emulator's locale from the Device Selector.

1. Right-click on a device and choose Properties.
2. In the Properties window, find the Locale property and click ... to open the Locale window.
3. Select the locale from the dropdown list.

Alternatively, while the emulator is running, select Application > Change Locale and type in the locale you want to use.

## Using the Resource Manager

To launch the resource manager, select a project, then choose Tools > i18n Resources Manager.

All the resources for the selected project are displayed in the Resource Manager. See the sample project i18nDemo described in "Running i18nDemo" on page 229.

See also: Working With Locales and Working With Resource Files.

## Working With Locales

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

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

- To add a locale, right-click on the top-level `global` node and choose Add Locale. Choose the locale from the combo box, or type it directly, and click OK.
- To rename a locale, right-click the locale directory and choose Rename.
- To remove a locale and all its contained resource files, right-click the locale directory and choose Delete.

## Working With Resource Files

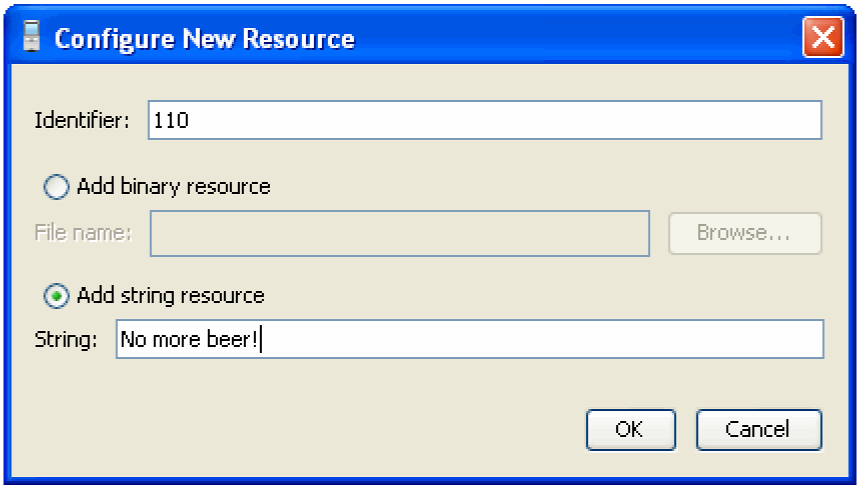
Resource files can be global (at the top level) or specific to a locale.

- To create a new global resource file, right-click the top-level `global` directory and choose Add new resource file. Choose a name for the file.
- To rename a resource file, right-click the file and choose Rename.
- You can copy, cut, and paste entire resource files. Right-click a file and choose Copy or Cut. Then right-click the locale directory (or the top-level `global`) and choose Paste.
- To remove a resource file, right-click the file and choose Delete.

## Working With Resources

Click on a resource file to display its contents.

- To add an image or another type of binary data, click the Add button.
  - In the Configure New Resource window, select Add string resource.
  - Browse to select the file you want to add.
  - The automatically supplied Identifier value can be changed.
  - Click OK to add the resource.



Identifier: 110

[✓ Add string resource]

String: No more beer!

- To edit a resource, double-click in the resource field.
  - For strings you can edit an existing value.
  - Double-clicking a binary file opens a file chooser.

# Running i18nDemo

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

---

**Note –** The default fonts for the Java ME Platform SDK do not support Chinese and Japanese. To use these languages, follow these steps before running this demo:

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

---

To run a MIDlet, highlight the MIDlet, then use the Launch soft button to run the MIDlet.

**String Comparator**

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

**Formatter**

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

**MicroLexicon**

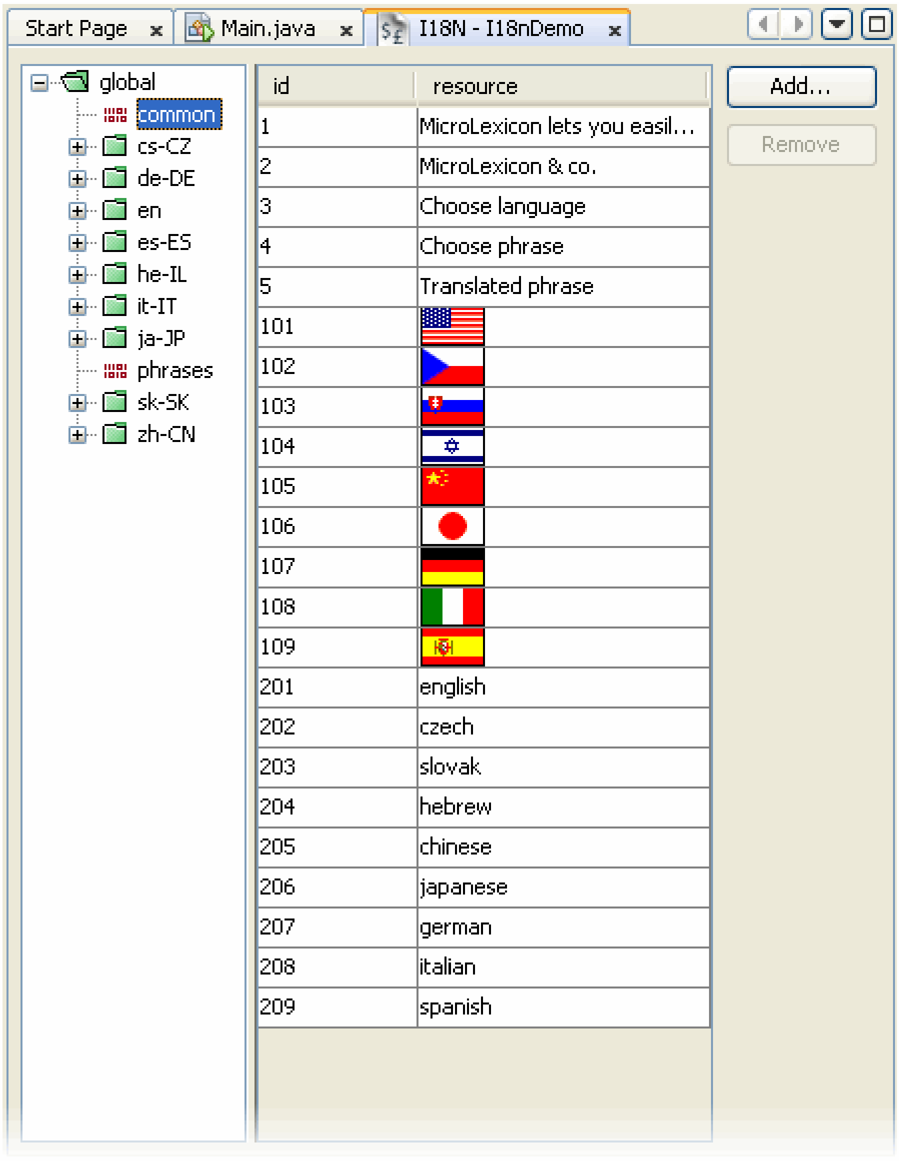
The final MIDlet, MicroLexicon, translates phrases from one language to another language. To set the source language, follow the steps in Setting the Emulator's Locale.

To select the target language from the list, use the navigation arrows to highlight Choose Language. Click the Select button to view the language drop down. Use the navigation arrows to choose a language and then click Select. Click the Next soft button.

MicroLexicon displays a list of phrases. Highlight one and press the Select button on the emulator.

MicroLexicon displays the flag of the target language and the translated phrase.

MicroLexicon is powered by MIDlet resources. To understand how you can use the Java ME Platform SDK to localize an application, choose Tools > i18n Resources Manager. All the resources, both text and images, used by MicroLexicon, appear. You can edit the resources and run MicroLexicon again to see what happens.



| id | resource |
|---|---|
| 1 | MicroLexicon lets you easil... |
| 2 | MicroLexicon & co. |
| 3 | Choose language |
| 4 | Choose phrase |
| 5 | Translated phrase |
| 101 | 🇺🇸 |
| 102 | 🇨🇿 |
| 103 | 🇸🇰 |
| 104 | 🇮🇱 |
| 105 | 🇨🇳 |
| 106 | 🇯🇵 |
| 107 | 🇩🇪 |
| 108 | 🇮🇹 |
| 109 | 🇪🇸 |
| 201 | english |
| 202 | czech |
| 203 | slovak |
| 204 | hebrew |
| 205 | chinese |
| 206 | japanese |
| 207 | german |
| 208 | italian |
| 209 | spanish |

To practice creating and editing resources, see Working With Resource Files.

The resources are stored in the project's JAR file.

CHAPTER **30**

# JSR 229: Payment API Support

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

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

Because the Payment API is closely tied to provisioning and external device payment mechanisms, and because payments can only succeed in a trusted protection domain, always test and debug your Payment API applications using the Run via OTA feature.

- Running the Payment Console
- Running JBricks

## Running the Payment Console

The Payment Console is a simple monitoring tool that displays payment related transactions sent from a mobile application using the Payment API (JSR 229). The payment console monitors Payment Update File requests and Premium Priced SMS payments.

The Payment Console is implemented as an Http server running within the Device Manager process. The root for the Http server is *installdir*`/apps`.

---

**Note –** The Device Manager must be running before you launch the Payment Console.

---

To launch the Payment Console, select Tools > Payment Console.

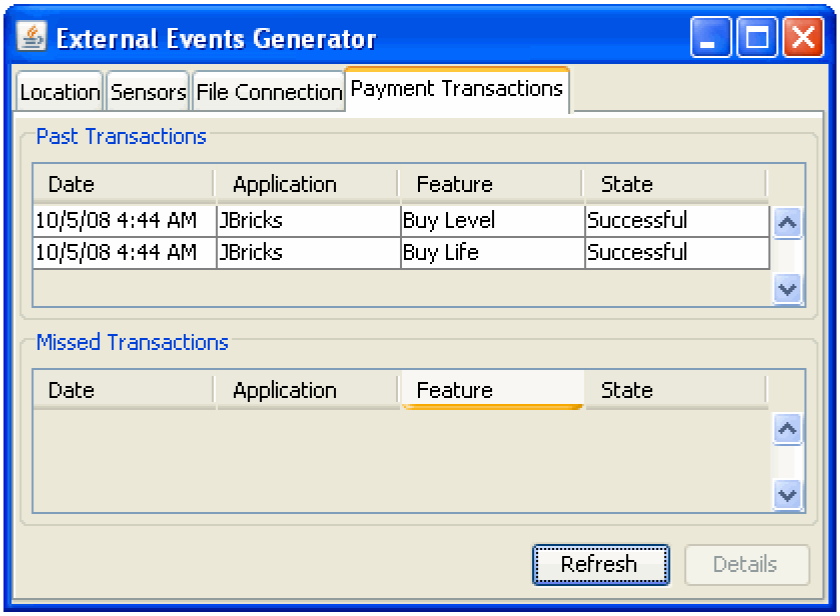
To close the Payment Console, right-click anywhere in the console and choose Close.

## Running JBricks

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

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

To view your transactions in the emulator, select View > External Events Generator and click on the Payment Transactions tab. Transactions for this specific instance of the emulator are displayed.



Past Transactions

| Date | Application | Feature | State |
|---|---|---|---|
| 10/5/08 4:44 AM | JBricks | Buy Level | Successful |
| 10/5/08 4:44 AM | JBricks | Buy Life | Successful |

Missed Transactions

| Date | Application | Feature | State |
|---|---|---|---|

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

```
PSP Console running, using phone number +5550001.
PSP Server running at https://localhost:-1
Received Payment Request from 127.0.0.1
   Credit card issued by: VISA
   Credit Card type: 0
   Credit Card Number: 4111111111111111
   Credit Card Holder: Jonathan Knudsen
   Feature ID: 3_lives
   Credit Card Verification Number (CCV): 123
   Payload: null
Response to 127.0.0.1
HTTP/1.1 200 OK
Content-Length: 0
Pay-Response: SUCCESSFUL
Pay-Timestamp: 1156282954734
```

CHAPTER 31

# 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 allows you to run a script that simulates sensor data.

You can use the custom API available with the SDK to create a custom sensor implementation with additional capabilities and support for different connection types.

## Creating a Mobile Sensor Project

To use this API, create a project with the target platform Custom. You must select MIDP 2.0 or higher and CLDC 1.1 before you can select the Mobile Sensor API optional package.

To set permissions, click the Settings button and choose the Permissions icon.

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.

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
```

## Using a Mobile Sensor Project

A Sensor project can be installed over the air. In the emulator window, select View > External Events Generator, and select the Sensors tab. 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 script player control that allows you to play sensor value events from a script file of the format discussed in Creating a Sensor Script File. See Running the Sensors Sample Project for a description of how to use the Sensors demo.

## 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>
```

The file *installdir*`/apps/Sensors/marbles.xml` 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 pre-configured sensors provided with this release are embedded, so they cannot be deactivated. If you configure your own sensor that is not embedded, it will be possible to deactivate it.

## Running the Sensors Sample Project

The Sensors demonstration has two MIDlets, SensorBrowser and Marbles that demonstrate the SDK’s implementation of the Mobile Sensor API.

Use the Run via OTA feature to install the application into the emulator.

## SensorBrowser

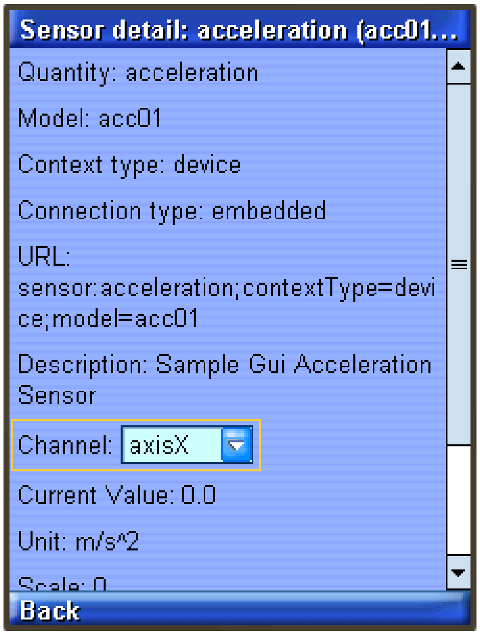
The SensorBrowser application displays the sensor detail information for each channel defined for the demo.

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

   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 (acc01...

Quantity: acceleration

Model: acc01

Context type: device

Connection type: embedded

URL:
sensor:acceleration;contextType=device;model=acc01

Description: Sample Gui Acceleration Sensor

Channel: axisX

Current Value: 0.0

Unit: m/s^2

Scale: 0

Click Back, then click Exit to return to the application menu.

## 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 View > External Events Generator, and select the Sensors tab.

   The emulator displays a list of the sensors in this application.

3. Click the Browse button and choose the following file: *installdir*`/apps/Sensors/marbles.xml`.

4. 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.

# Index