Oracle® Java ME Embedded

Getting Started Guide for the Reference Platform (Raspberry Pi)

Release 8.3

E73089-02

July 2016

This guide describes how to install and run the Oracle Java ME Embedded software on the Raspberry Pi reference platform.

Oracle Java ME Embedded Getting Started Guide for the Reference Platform (Raspberry Pi), Release 8.3

E73089-02

Contents

Preface vii

Audience vii
Related Documents vii
Shell Prompts vii
Conventions vii

1 Installing Oracle Java ME Embedded Software on the Raspberry Pi Board

Required Hardware and Software Items 1-1
Installing the Raspbian Pi Operating System on the Raspberry Pi Board 1-2
Preparing the Raspberry Pi Board for Work 1-2
Preparing Raspberry Pi Board for Working With an Embedded Display 1-3
Attaching the Embedded Display to the Raspberry Pi Board 1-3
Installing the Customized Operating System Image on the Raspberry Pi Board 1-3
Installing the Oracle Java ME Embedded Software 1-4
Structure of the Oracle Java ME Embedded Software 1-4
Installing the Oracle Java ME Embedded Software Using the Java ME SDK 1-4
Installing the Oracle Java ME Embedded Software Manually 1-6
Adding an HTTP Proxy for Network Connections on Raspberry Pi 1-7
Downloading and Installing the PuTTY Terminal Emulator Program 1-7

2 Installing and Running Applications on the Raspberry Pi Board

Ways of Using the Java Runtime on the Raspberry Pi Board 2-1
Preinstalled IMlets 2-2
Run of IMlets on Raspberry Pi Using the Command Shell 2-2
An Example of Managing Application Life Cycle with Shell Commands 2-2
Purpose of the Developer Agent Program on the Desktop 2-3
Configuring Client Mode Connection 2-3
Installing the Developer Agent Program 2-4
Starting the Developer Agent in a Server Mode 2-4
Starting the Developer Agent in a Client Mode 2-4
Secured Connection to the Developer Agent 2-5
Generating the Developer Agent Connection CA Certificate 2-5

Installing the Developer Agent CA Certificate on the Device 2-5
Configuring the Java Runtime Properties 2-5
Running the Developer Agent.... 2-5
Running IMlets on Raspberry Pi Using the AMS CLI.... 2-6
Making a Raw Connection to the AMS CLI 2-6
Lists of Commands.... 2-8
An Example of Managing Application Life Cycle with AMS Commands.... 2-8
NetBeans and the Raspberry Pi Board.... 2-9
Required Software for Using the Raspberry Pi Board with NetBeans 2-9
Adding the Raspberry Pi Board to the Device Connection Manager.... 2-9
Assigning the Raspberry Pi Board to Your Project 2-10
Creating and Running Java ME Application in NetBeans 2-11
Debugging an IMlet on the Raspberry Pi Board.... 2-12
Access to Peripherals.... 2-12
Signing the Application with API Permissions 2-13
Activating Stronger Signature Verification 2-18
Obtaining Java Logs from a Device.... 2-18
Obtaining Java Logs Via an SDK Output Console Window.... 2-18
Obtaining Java Logs Via a Console Application.... 2-20
List of Logging Parameters in the jwc_properties.ini File 2-21
Work With an Embedded Display 2-22
Setting Up a Primary Display.... 2-22
The Embedded Display ID.... 2-22

3 Troubleshooting

Installing Linux on the Raspberry Pi Board.... 3-1
Starting Oracle Java ME Embedded Software on the Board 3-1
Using the Board with the Oracle Java ME SDK and the NetBeans IDE 3-2

A Device I/O Preconfigured List

GPIO Pins.... A-1
I2C A-5
MMIO A-6
SPI A-6
UART.... A-7
Watchdog A-8
ATCmdDevice.... A-8
Structure of the Device I/O Configuration File A-8

B Configuring the Java Runtime Properties

Editing the jwc_properties.ini File B-1
Using the CLI set-property Command.... B-1

Glossary

Preface

This guide describes how to install Oracle Java ME Embedded software onto a Raspberry Pi device.

Audience

This guide is for developers who want to run Oracle Java ME Embedded software on a Raspberry Pi device.

Related Documents

For a complete list of documents for the Oracle Java ME Embedded software, see the *Release Notes*.

Shell Prompts

Shell	Prompt
Windows	*directory*>
Linux	$

Conventions

The following text conventions are used in this guide:

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

1

Installing Oracle Java ME Embedded Software on the Raspberry Pi Board

This chapter describes installing the Oracle Java ME Embedded 8.3 software on the Raspberry Pi board, installing and using the Developer Agent program on the desktop host, configuring the Oracle Java ME Embedded system, and connecting to the Raspberry Pi using a secure shell.

This chapter contains the following topics:

- Required Hardware and Software Items
- Installing the Raspbian Pi Operating System on the Raspberry Pi Board
- Preparing the Raspberry Pi Board for Work
- Preparing Raspberry Pi Board for Working With an Embedded Display
- Structure of the Oracle Java ME Embedded Software
- Installing the Oracle Java ME Embedded Software Using the Java ME SDK
- Installing the Oracle Java ME Embedded Software Manually
- Adding an HTTP Proxy for Network Connections on Raspberry Pi
- Downloading and Installing the PuTTY Terminal Emulator Program

Required Hardware and Software Items

This section describes the hardware and software items that are required for developing on the Raspberry Pi board.

- Raspberry Pi 2, Raspberry Pi Rev. B or Rev. B+ 512 MB board
- Oracle Java ME Embedded 8.3 distribution or Oracle Java ME SDK 8.3

Note:

You have two options to work with the Raspberry Pi board: with or without using the Java ME SDK. You can use the Java ME SDK to setup your Raspberry Pi device or download the Oracle Java ME Embedded distribution and install it manually.

- Micro-USB power supply of 0.7 or greater amps, and 5 volts. Note that the power supply must have a *micro*-USB type B connector, not a regular USB or mini-USB connector.

- USB keyboard, mouse, and a monitor. If necessary for your monitor, use an HDMI-to-DVI video cable or adapter.
- SD card of 4 GB or greater. An SD-HC class 10 card is recommended. Do not use a high-speed SD card, because it may be too fast for the Raspberry Pi board.
- Ethernet cable with an RJ-45 connector, as well as a connection to a network with a DHCP server.
- A terminal emulator program, such as PuTTY, if you wish to connect to the board using the Application Management System (AMS) interface.

Note:

Oracle Java ME 8.3 supports working with any embedded display with the frame buffer interface such as Adafruit PiTFT 3.5" Touch Screen for Raspberry Pi.

Installing the Raspbian Pi Operating System on the Raspberry Pi Board

To develop applications on the Raspberry Pi board, you must first download and install the Raspbian operating system (based on Debian Wheezy) on the Raspberry Pi board. Note that if you want to use the embedded display, you must download the customized operating system image. For more information, see Preparing Raspberry Pi Board for Working With an Embedded Display.

To install the Raspbian Pi operating system, follow these steps:

1. Download the Raspbian raw image ZIP file to your desktop from the following site:

   `http://www.raspberrypi.org/downloads`

2. Unzip the distribution file, which creates a single disk image (`.img`) file.

3. Mount the SD card to the desktop, and use a utility to write the disk image file to the SD card. Note that this is not the same as copying the file to the base-level directory on the SD card. Instead, it is similar to *burning* a disk image onto a CD-ROM or DVD-ROM. There are a number of utilities that perform this action:

   - For the Windows operating system, you can use the Disk Image Writer utility located at `https://launchpad.net/win32-image-writer`.
   - For the Mac platform, use the RPi-sd Card Builder located at `http://alltheware.wordpress.com/2012/12/11/easiest-way-sd-card-setup`.
   - For Linux, use the `dd` command. For more information, see `http://en.wikipedia.org/wiki/Dd_(Unix)`.

4. Eject or unmount the SD card from the desktop computer.

5. Install the SD card in the Raspberry Pi board.

Preparing the Raspberry Pi Board for Work

To prepare the Raspberry Pi board for work, follow these steps:

1. Connect the RJ-45 network cable, monitor, keyboard, and mouse to your Raspberry Pi board.
2. Connect power to the Raspberry Pi board. The red light on the Raspberry Pi board should glow, then in a few seconds, the green light should blink. The blinking green light indicates that the Raspberry Pi board is booting Linux.
3. If the Linux installation was successful, the Raspberry Pi board will start and obtain a DHCP address.
4. A configuration program (`raspi-config`) runs, which helps you expand the file system partition on the SD card, configure the keyboard and time zone, reset the default password, and so on. Use the up and down arrow keys to make a menu choice. Use the left or right arrow keys to select **OK** or **Cancel**. Press `Return` to run your choice. Note that the default user name is `pi`, and the default password is `raspberry`.
5. You can perform an update, start the `ssh` server, and set the graphical desktop to automatically start, then click **Finish**. At this point, the board should reboot.
6. Log in if necessary, and if you are using the desktop, start an LXTerminal.
7. Run the `ifconfig` command to display the Raspberry Pi IP address. This is necessary so that you can access and control the board remotely. Remember this IP address for further use.

Preparing Raspberry Pi Board for Working With an Embedded Display

Oracle Java ME 8.3 supports working with any embedded display with the frame buffer interface such as Adafruit PiTFT 3.5" Touch Screen for Raspberry Pi.

This section explains how to prepare your board for working with the embedded display and contains the following topics:

- Attaching the Embedded Display to the Raspberry Pi Board
- Installing the Customized Operating System Image on the Raspberry Pi Board
- Installing the Oracle Java ME Embedded Software

Attaching the Embedded Display to the Raspberry Pi Board

Attach the embedded display using the high speed SPI interface on the Raspberry Pi board. For more information, see the manufacturer's site at

`https://learn.adafruit.com/adafruit-pitft-3-dot-5-touch-screen-for-raspberry-pi/overview`

Installing the Customized Operating System Image on the Raspberry Pi Board

Download the customized Raspbian based raw image ZIP file to your desktop from the following site:

`https://learn.adafruit.com/adafruit-pitft-3-dot-5-touch-screen-for-raspberry-pi/easy-install`

Install the image into the Raspberry Pi board as described in Installing the Raspbian Pi Operating System on the Raspberry Pi Board.

Installing the Oracle Java ME Embedded Software

After you installed the customized operating system image, you must re-install the Java ME Embedded software using one of the following procedures:

- Installing the Oracle Java ME Embedded Software Using the Java ME SDK
- Installing the Oracle Java ME Embedded Software Manually

Structure of the Oracle Java ME Embedded Software

Starting from version 8, the Oracle Java ME Embedded software contains a different architecture than previous versions. The user has an option to run a Developer Agent program on the desktop under Windows or Linux. Commands that are sent to the board from the host desktop are no longer sent directly across the network. Instead, they are sent to the Developer Agent program, which transmits all communication to and from the Oracle Java ME Embedded executable file on the Raspberry Pi board.

The Oracle Java ME Embedded ZIP archive consists of the following directories:

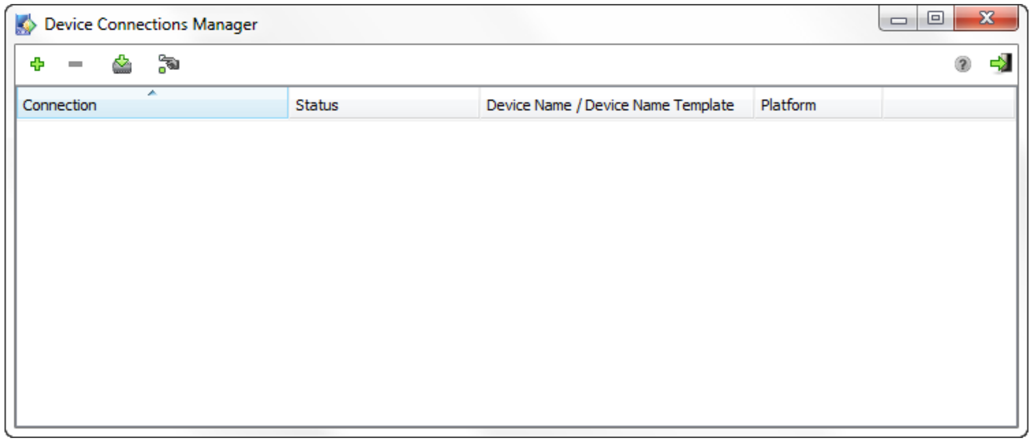
- `/appdb:` This directory is used on the Raspberry Pi and contains internal Java libraries.
- `/bin`: This directory is used on the Raspberry Pi and contains executables and the `jwc_properties.ini` file.
- `/legal:` This directory contains important legal documentation.
- `/lib`: This directory contains the files needed to compile IMlets on the Raspberry Pi board.
- `/util:` This directory contains the Developer Agent program.

Installing the Oracle Java ME Embedded Software Using the Java ME SDK

This section describes how to install the Oracle Java ME Embedded Software on the Raspberry Pi board using the Oracle Java ME SDK. If you want to manually install the Java Embedded Software, see Installing the Oracle Java ME Embedded Software Manually.

1. Install the Oracle Java ME SDK 8.3. For complete instructions, see *Oracle Java ME SDK Developer's Guide*.
2. Start the Oracle Java ME SDK Device Connections Manager located in the <*SDK Installation Folder*>`/bin` directory and click its icon in the system tray. A Device Connections Manager window is shown in Figure 1-1.

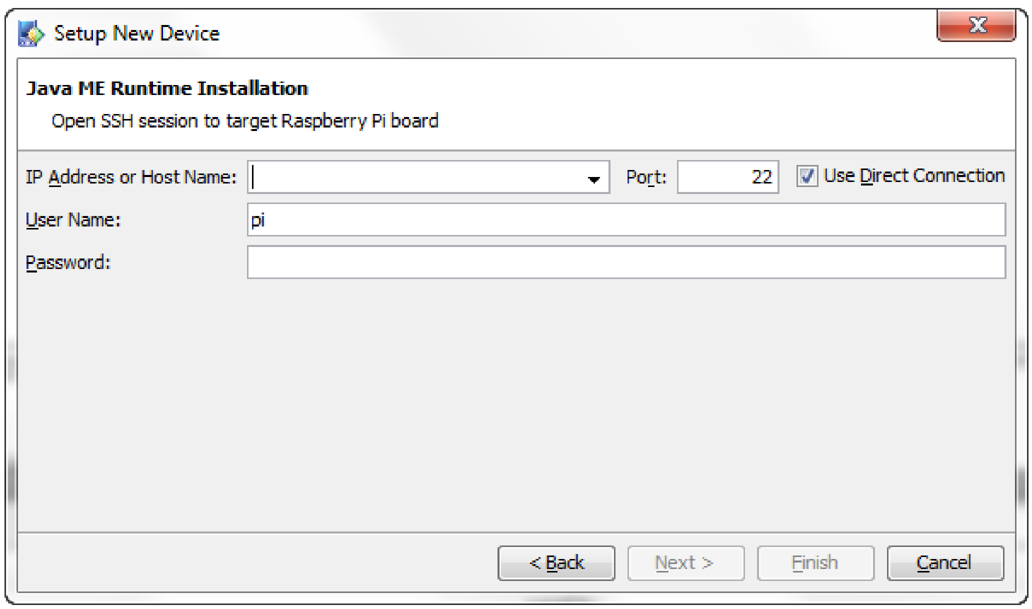
Figure 1-1 Device Connections Manager Window



3. Press **Ctrl+N** or click the Setup new device button on the toolbar.

4. Select the Raspberry Pi radio button and click **Next**. The Setup New Device window is shown in Figure 1-2.

Figure 1-2 Setup New Device Window



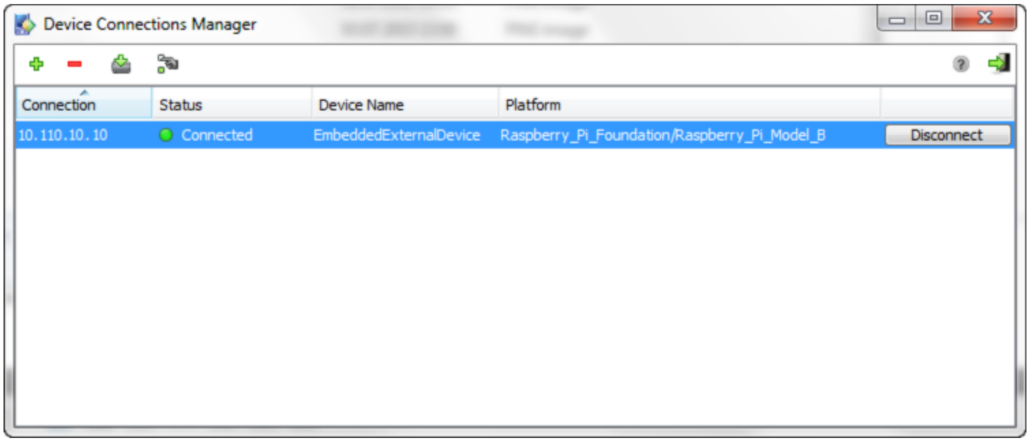
5. In the IP Address or Host Name, enter the IP address of the Raspberry Pi board, in the User Name and Password fields, enter the user name and password. Click **Next.**

6. Choose the installation folder. By default, the Java ME Runtime is installed in the `/home/pi/javame8.3` directory.

7. Check the Start Java ME Runtime Now and Start Java ME Runtime on Boot check boxes and click **Finish**.

The Raspberry Pi device is now registered and has the Connected status in the Device Connections Manager window as shown in Figure 1-3.

Figure 1-3 Device Connections Manager Window with Raspberry Pi Connected



Installing the Oracle Java ME Embedded Software Manually

This section describes how to install the Oracle Java ME Embedded Software manually. Download the Oracle Java ME Embedded ZIP archive file and follow these steps:

1. Use an `sftp` client or `scp` command to transfer a copy of the Oracle Java ME Embedded archive to the Raspberry Pi board.

   For example, on a UNIX or Mac system, you can transfer the ZIP file using a command similar to the following:

   ```
   $sftp pi@[IP address of board]
   ```

   Windows users can download the `psftp.exe` to obtain a free SFTP client which is available from the same address as the PuTTY executable:

   `http://www.putty.org/`

2. After the ZIP archive is transferred, either go directly to the keyboard and the mouse connected to the Raspberry Pi board, or start a secure shell script on your desktop to connect to the board using the following command:

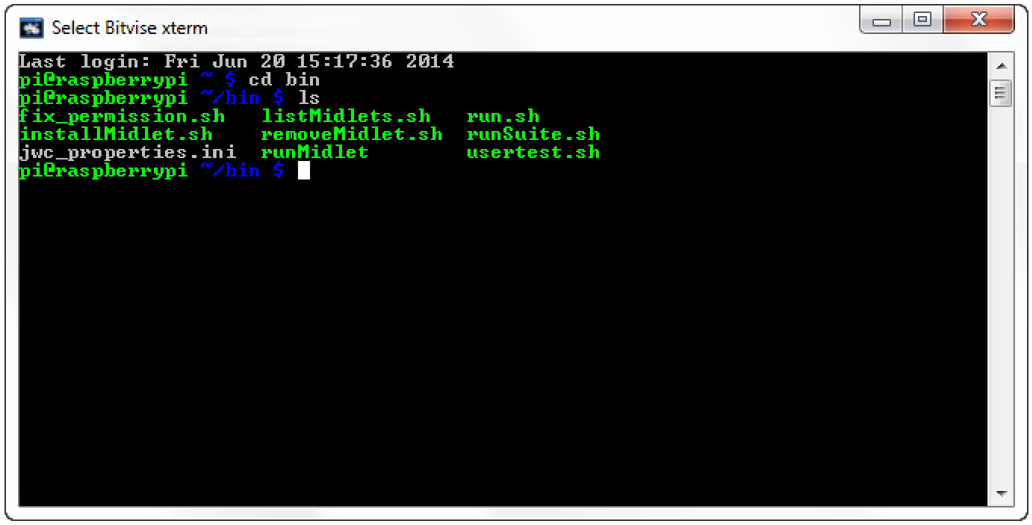
   ```
   $ssh -l pi [IP address of board]
   ```

3. Unzip the archive on the Raspberry Pi board.

4. Change to the `bin` directory.

   The contents of the `bin` directory are shown in Figure 1-4.

Figure 1-4 Raspberry Pi Bin Directory



5. Run the `fix_permission.sh` script to enable a privilege escalation mechanism.

```
pi@raspberry ~/bin $ fix_permission.sh
```

The `fix_permission.sh` provides the following functionality:

- Sets an executable mode for all `.sh` files
- Sets an executable mode for the `runMidlet`
- Sets the network capabilities for the `runMidlet` by granting superuser privileges to perform various network-related operations
- Enables the `runMidlet` to access the files required for the Device I/O functionality

Adding an HTTP Proxy for Network Connections on Raspberry Pi

If an HTTP proxy server is required for the Java IMlets on Raspberry Pi to make network connections (such as for HTTP or `apt-get`), then configure Oracle Java ME Embedded on Raspberry Pi by adding the following lines to the end of the `bin/jwc_properties.ini` file:

```
com.sun.midp.io.http.proxy.host = proxy.mycompany.com
com.sun.midp.io.http.proxy.port = 80
```

Downloading and Installing the PuTTY Terminal Emulator Program

The PuTTY terminal emulator is used to connect to the AMS command-line interface (CLI) that sends commands to the board.

Download the PuTTY terminal emulator program (`putty.exe`) from the following site:

`http://www.putty.org/`

Note:

Using the PuTTY terminal emulator program is highly recommended. You can use any terminal program to connect to the CLI, however, Oracle cannot guarantee that other terminal programs work with the CLI in the same manner as PuTTY.

2

Installing and Running Applications on the Raspberry Pi Board

This chapter describes how to run IMlets using the command-line shell interface, the Application Management System (AMS) command-line interface (CLI), and NetBeans IDE. You learn how to add the board to the Device Connections Manager in the Oracle Java ME SDK and how to debug an IMlet on the board from the NetBeans IDE.

Topics:

- Ways of Using the Java Runtime on the Raspberry Pi Board
- Preinstalled IMlets
- Run of IMlets on Raspberry Pi Using the Command Shell
- An Example of Managing Application Life Cycle with Shell Commands
- Purpose of the Developer Agent Program on the Desktop
- Installing the Developer Agent Program
- Starting the Developer Agent in a Server Mode
- Starting the Developer Agent in a Client Mode
- Secured Connection to the Developer Agent
- Running IMlets on Raspberry Pi Using the AMS CLI
- NetBeans and the Raspberry Pi Board
- Access to Peripherals
- Obtaining Java Logs from a Device
- List of Logging Parameters in the jwc_properties.ini File
- Work With an Embedded Display

Ways of Using the Java Runtime on the Raspberry Pi Board

There are several ways to use the Oracle Java ME Embedded platform on the Raspberry Pi board.

- Directly run commands using a command-line shell interface or logging in using the `ssh` protocol.
- Manually start a Developer Agent program on the desktop host and run commands using the Application Management System (AMS).

- Run NetBeans IDE.

Preinstalled IMlets

The Java ME Embedded runtime delivery might include preinstalled IMlets. When started, the Java ME runtime checks availability of preinstalled IMlets and automatically installs them on the Raspberry Pi board, if they were not installed during previous runs.

Preinstalled IMlet files are located in the `/appdb` directory on the board and have the following format:

- `/appdb/Preinstalled_app_<num>.jar`

  A mandatory file that contains application classes.

- `/appdb/Preinstalled_app_<num>.jad`

  An optional file that contains an application descriptor.

In this notation, `<num>` stands for a preinstalled IMlet sequence number. Can be any number starting from 1.

Run of IMlets on Raspberry Pi Using the Command Shell

You can run IMlets directly on the Raspberry Pi board by using the commands shown in Table 2-1.

Table 2-1 Raspberry PiShell Commands

Syntax	Description
`listMidlets.sh [SUITE_ID or NAME]`	List all installed IMlet suites and their status or show the detail of a single suite.
`installMidlet.sh <URL> [<URL label>]`	Install an IMlet using the specified JAR file.
`removeMidlet.sh <SUITE_ID>`	Remove an installed IMlet.
`runSuite.sh <SUITE_ID or NAME> [IMLET_ID or classname]`	Run the specified IMlet or the default if none is specified. All logging information from the IMlet appears in the standard output of this command.

Note:

The term *IMlet*, in the context of theOracle Java ME Embedded command-line interface (CLI) and references in this chapter, is synonymous with *MIDlet*.

An Example of Managing Application Life Cycle with Shell Commands

The following is a typical example of using the commands to install, list, run, and remove an Oracle Java ME Embedded application on the Raspberry Pi Pi board. Most commands can be terminated with the Ctrl-C key combination if they become unresponsive.

First, install the application using the `installMidlet.sh` command, specifying its location on the local file system.

```
pi@raspberrypi ~/bin $ ./installMidlet.sh /home/pi/EmbeddedTestProject.jar
Java is starting. Press Ctrl+C to exit
The suite was successfully installed, ID: 2
```

After an IMlet is installed, note its ID: in this case, it is 2. Next, verify it using the `listMidlets.sh` command.

```
pi@raspberrypi ~/bin $ ./listMidlets.sh
Java is starting. Press Ctrl-C to exit
Suite: 2
  Name: EmbeddedTestProject
  Version: 1.0
  Vendor: Vendor
  MIDlets:
    MIDlet: GPIODemo
```

You can run any installed IMlet using the `runSuite.sh` command. This command runs the IMlet that was just installed, passing any logging information to the standard output of this command. Note that you can press the Ctrl-C key to exit from this command, which will terminate the application.

```
pi@raspberrypi ~/bin $ ./runSuite.sh 2
Java is starting. Press Ctrl-C to exit
Starting - GPIODemo
```

You can use the `removeMidlet.sh` command to remove any installed IMlet.

```
pi@raspberrypi ~/bin $ ./removeMidlet.sh 2
Java is starting. Press Ctrl-C to exit
Suite removed
pi@raspberrypi ~/bin $
```

You can verify the results by using the `listMidlets.sh` command.

```
pi@raspberrypi ~/bin $ ./listMidlets.sh
Java is starting. Press Ctrl-C to exit
No suites installed
```

Purpose of the Developer Agent Program on the Desktop

The Developer Agent program runs on the Windows or Linux desktop and transmits all communication to and from the Oracle Java ME Embedded executable file on the Raspberry Pi board.

You can start the Developer Agent program on the desktop host computer either in a server or a client mode. The server mode is used by default. The client mode must be configured in the `jwc_properties.ini` file.

Configuring Client Mode Connection

To configure a client mode connection, edit the `jwc_properties.ini` file in the `bin` directory on the Raspberry Pi board as follows.

```
proxy_connection_mode = client
proxy.client_connection_address = <IP address>
```

where <IP address> is the IP address of the desktop host computer running the Developer Agent program.

Installing the Developer Agent Program

The Developer Agent program is the proxy.jar file inside the `util` directory of the Oracle Java ME Embedded distribution.

To install the Developer Agent program, follow these steps:

1. Extract files from the copy of the Oracle Java ME Embedded ZIP archive on the host desktop.
2. Delete the `/appdb` and `/bin` directories.

Starting the Developer Agent in a Server Mode

When using the Developer Agent in a server mode, it is important that you first started the Java runtime on the Raspberry Pi board to allow access to the AMS.

To start the Developer Agent program, follow these steps:

1. Change to the `bin` directory on the Raspberry Pi board and run the `./usertest.sh` command:

   ```
   pi@raspberry ~ /pi/bin $ ./usertest.sh
   ```

2. Change to the `util` directory on your desktop host and enter the following command.

   ```
   C:\mydir\util> java -jar proxy.jar -socket <RPI IP ADDRESS>
   ```

   You should see an output similar to the following:

   ```
   Trying to open socket connection with device: <IP Address>:2201
   Connected to the socket Socket[addr=/<IP address>, port=2201, localport=54784]
   Open channel 8 with hash 0x390df07e
   notifyResponse AVAILABLE_RESPONSE on channel 8
   Channel 8 CLOSED -> AVAILABLE
   Open channel 9 with hash 0x0
   ```

After the Developer Agent starts, use the AMS CLI.

Starting the Developer Agent in a Client Mode

When using the Developer Agent in a client mode, ensure that you configured the client mode as described in the Configuring Client Mode Connection.

To start the Developer Agent program, use these steps:

1. Run the `usertest.sh` command in the `/bin` directory:

   ```
   pi@raspberry ~ /bin $ ./usertest.sh
   ```

2. Change to the `util` directory on your desktop host and enter the following command.

   ```
   C:\mydir\util> java -jar proxy.jar
   ```

   You should see an output similar to the following:

```
Starting with default parameters: -ServerSocketPort 2200 -jdbport 2801
Channel 8 CLOSED -> AVAILABLE
Waiting for device connections on port 2200
```

After the Developer Agent starts, use the AMS CLI.

Secured Connection to the Developer Agent

The Oracle Java ME Embedded software enables you to set up a secured channel between the device and the Developer Agent so that the commands are sent over the TLS-encrypted channel. This section contains the following topics:

- Generating the Developer Agent Connection CA Certificate
- Installing the Developer Agent CA Certificate on the Device
- Configuring the Java Runtime Properties
- Running the Developer Agent

Generating the Developer Agent Connection CA Certificate

Generate the Developer Agent connection CA certificate by performing the Java SE keytool command:

```
keytool.exe -genkeypair -keystore <JKS keystore file> -alias
<keypair alias> -keyalg RSA -keysize 4096 -validity 7305 -
storepass <keystore password> -keypass <keypair password>
```

Installing the Developer Agent CA Certificate on the Device

Install the Developer Agent CA Certificate on your device by performing the CLI command while the device and Developer Agent are being connected:

```
ks-import -proxy -keystore <JKS keystore file> -storepass
<keystore password> -keypass <keypair password> -alias <keypair
alias>
```

This command installs the CA certificate to the `/appdb/cert_proxy` directory.

Configuring the Java Runtime Properties

To configure the Java runtime properties involved in establishing a secured connection to the Developer Agent, set the following properties:

`proxy.certificate=`<CA certificate name>

`proxy.secured=true`

Running the Developer Agent

After the secured connection to the Developer Agent is configured, restart the Developer Agent using the following command:

```
java -jar proxy.jar -socket 127.0.0.1 -socketPort 51300 -
secureConnection -keystoreFile <JKS keystore file> -
keystorePassword <keystore password> -keypairAlias <keypair
alias> -keypairPassword <keypair password> -cliport 65002
```

Running IMlets on Raspberry Pi Using the AMS CLI

To run IMlets on the Raspberry Pi board using the AMS CLI, you must first make a raw connection to the AMS CLI and then use commands.

This section contains the following topics:

- Making a Raw Connection to the AMS CLI
- Lists of Commands
- An Example of Managing Application Life Cycle with AMS Commands

Making a Raw Connection to the AMS CLI

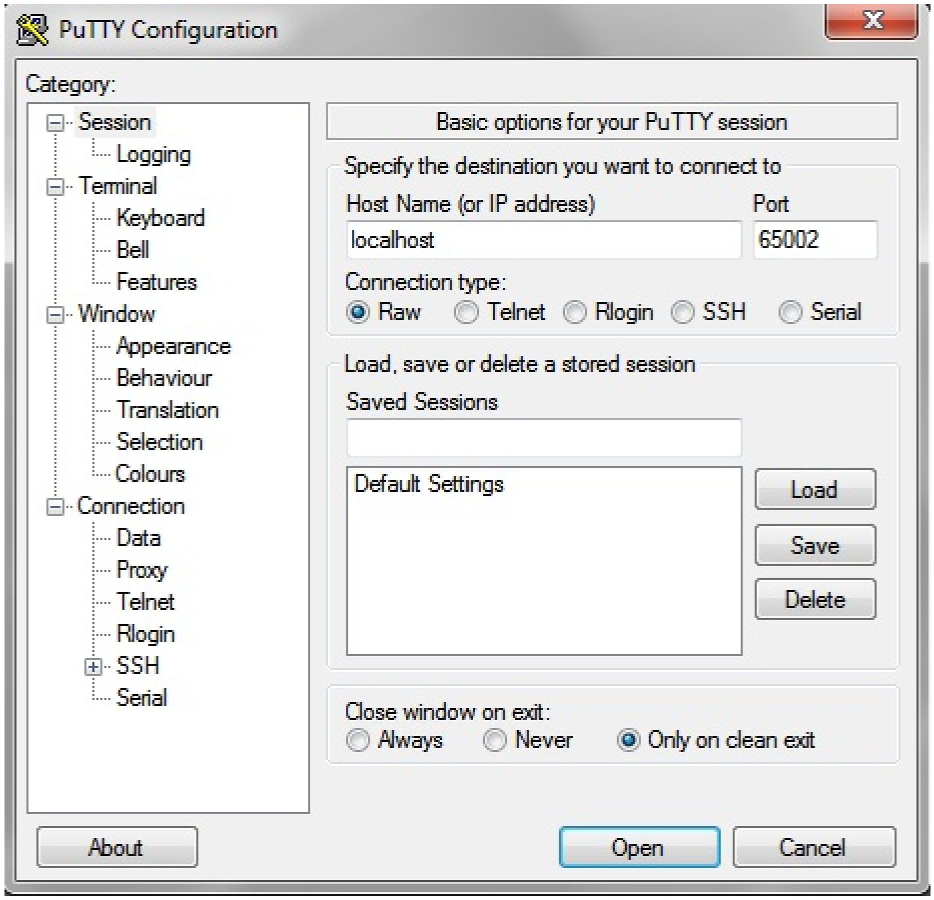
Before making a raw connection to the AMS CLI ensure that you started the Developer Agent program on the desktop host and the Java runtime - on the Raspberry Pi board unless the Developer Agent was started automatically by Java ME SDK.

To make a raw connection to the AMS CLI, perform the following.

1. Start a PuTTY executable file on your desktop computer.
2. Create raw socket connections to the IP address of the host running the Developer Agent, and port 65002.

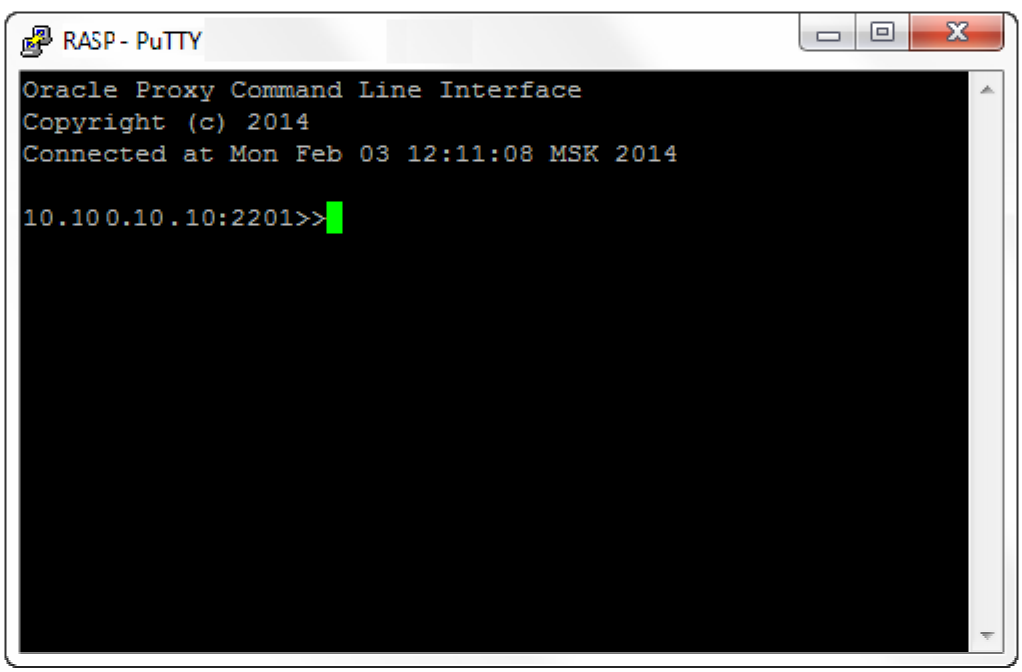
A connection to `localhost` and the port 65002 is shown in Figure 2-1.

Figure 2-1 Using PuTTY to Connect to the Command-Line Interface



The window from port 65002 provides a CLI as shown in Figure 2-2.

Figure 2-2 Command-Line Interface to Raspberry Pi



Note:

The CLI feature in this Oracle Java ME Embedded software release is provided only as a concept for your reference. It uses connections that are not secure, without encryption, authentication, or authorization.

Lists of Commands

For a complete list of CLI commands, see Using the Command Line Interface in *Oracle Java ME Embedded Developer's Guide*.

An Example of Managing Application Life Cycle with AMS Commands

Here is a typical example of using the AMS to install, list, run, and remove an Oracle Java ME Embedded application on the board:

```
oracle>> ams-install file:///C:/some/directory/hello.jar hostdownload
<<ams-install,start install,file:///C:/some/directory/hello.jar
<<ams-install,install status: stage DONE, 0%
<<ams-install,install status: stage DONE, 100%
<<ams-install,OK,Install success

oracle>> ams-install http://www.example.com/netdemo.jar
<<ams-install,start install,http://www.example.com/netdemo.jar
<<ams-install,install status: stage DONE, 0%
<<ams-install,install status: stage DONE, 100%
<<ams-install,OK,Install success

oracle>> ams-install http://www.example.com/notthere.jar
<<ams-install,start install,http://www.example.com/notthere.jar
<<ams-install,FAIL,errorCode=103 (OTHER_ERROR)
```

Note that the final installation example failed with an error code and matching description.

Similarly, install an additional IMlet: `rs232dem`. After an IMlet is installed, verify it using the `ams-list` command. Each IMlet has been assigned a number by the AMS for convenience.

```
oracle>> ams-list
<<ams-list,0.hello|Oracle,STOPPED
<<ams-list,1.netdemo|Oracle,STOPPED
<<ams-list,2.rs232dem|Oracle,RUNNING
<<ams-list,OK,3 suites are installed
```

You can use the `ams-remove` command to remove any installed IMlet.

```
oracle>> ams-remove 0
<<ams-remove,OK,hello removed
```

The results can again be verified with the `ams-list` command.

```
oracle>> ams-list
<<ams-list,1.netdemo|Oracle,STOPPED
<<ams-list,2.rs232dem|Oracle,RUNNING
<<ams-list,OK,2 suites are installed
```

Finally, start the IMlet using the `ams-run` command. The application can be terminated with the `ams-stop` command.

```
oracle>> ams-run 1
<<ams-run,OK,started

oracle>> ams-list
<<ams-list,1.netdemo|Oracle,RUNNING
<<ams-list,2.rs232dem|Oracle,RUNNING
<<ams-list,OK,2 suites are installed
```

NetBeans and the Raspberry Pi Board

Topics:

- Required Software for Using the Raspberry Pi Board with NetBeans
- Adding the Raspberry Pi Board to the Device Connection Manager
- Assigning the Raspberry Pi Board to Your Project
- Creating and Running Java ME Application in NetBeans
- Debugging an IMlet on the Raspberry Pi Board

Required Software for Using the Raspberry Pi Board with NetBeans

Running and debugging IMlet projects on the Raspberry Pi board using the NetBeans IDE 8.1 requires the following software:

- NetBeans IDE 8.1 with Java ME 8.3 support
- Oracle Java ME SDK 8.3
- Oracle Java ME SDK 8.3 plugins

For complete instructions about installing Oracle Java ME SDK 8.3, the NetBeans IDE 8.1, and Oracle Java ME SDK 8.3 plug-ins for NetBeans, see *Oracle Java ME SDK Developer's Guide*.

Note:

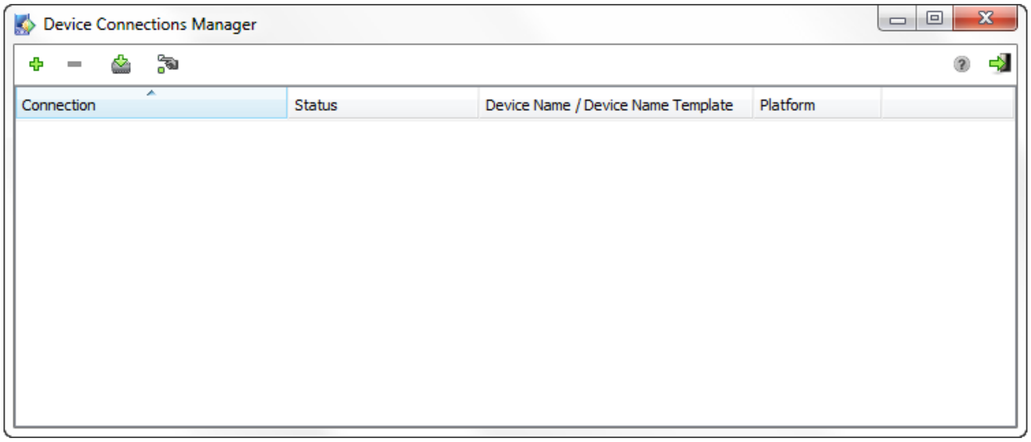
This chapter assumes that the Raspberry Pi board is already set up and connected to the Windows or Linux platform running Oracle Java ME SDK8.3 and that NetBeans IDE 8.1 has already been started.

Adding the Raspberry Pi Board to the Device Connection Manager

If you want to use the Raspberry Pi with NetBeans, you must first add the Raspberry Pi to the Device Connection Manager in Oracle Java ME SDK 8.3 as follows.

1. Ensure that the `usertest.sh` script in the `/bin` directory is running on the Raspberry Pi board.
2. Ensure that the Developer Agent program does not run on the desktop computer.
3. Start the Oracle Java ME SDK 8.3 Device Connections Manager (located in the <*SDK Installation Folder*>`/bin`) directory and click its icon in the system tray. A Device Connections Manager window is shown in Figure 2-3.

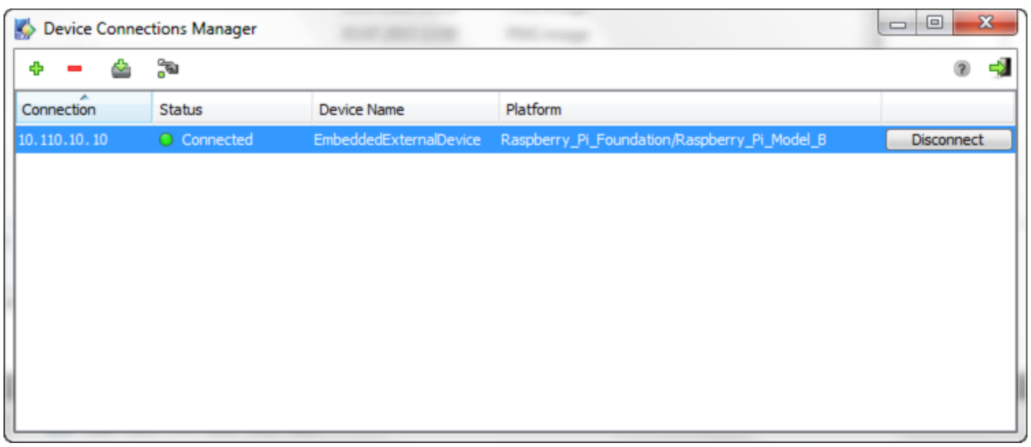
Figure 2-3 Device Connections Manager Window



4. Click the **Add new device connection** button, ensure that the IP Address or Host Name list contains the correct IP address of the Raspberry Pi board, and click **OK**.

5. After the Raspberry Pi board is registered, its IP address is listed on the Device Connections Manager list and its status is Connected as shown in Figure 2-4.

Figure 2-4 Device Connections Manager Window with Raspberry Pi Connected



Assigning the Raspberry Pi Board to Your Project

There are two ways to assign the Raspberry Pi board to your project:

- Using an Existing NetBeans Project
- Creating a New NetBeans Project

After you assign the board to your project, clicking **Run Project** in the NetBeans IDE runs your IMlet on the board instead of on the emulator.

Using an Existing NetBeans Project

If you already have an existing NetBeans project with an IMlet that you want to run or debug on the board, follow these steps:

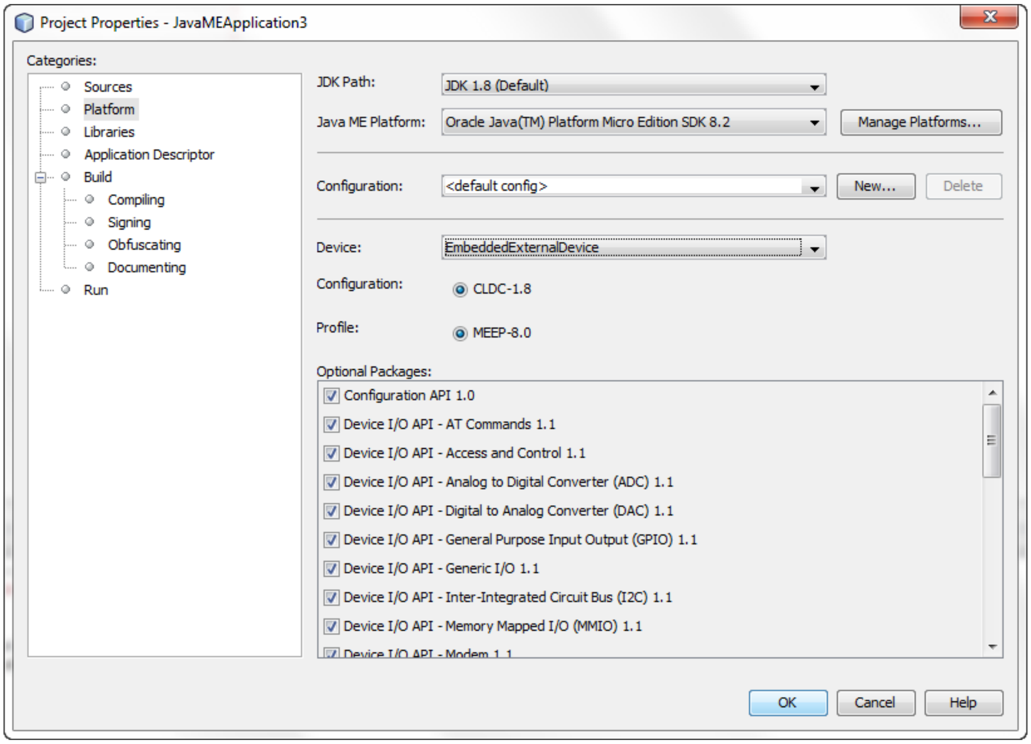
1. Right-click your project and select **Properties**.

2. Select the **Platform** category on the properties window.

3. Ensure that **Oracle Java(TM) Platform Micro Edition SDK 8.3** is selected in the Java ME Platform list.

4. Select **EmbeddedExternalDevice** from the Device drop-down list, as shown in Figure 2-5. Select (or deselect) from the list of Optional Packages as needed for your project, and click **OK**.

Figure 2-5 Adding a Device to Your Project



Creating a New NetBeans Project

Create a new NetBeans project with the name `ME8EmbeddedApplication1` and assign the Raspberry Pi board to it as described in Creating a Java ME Embedded Application Project in NetBeans IDE in the *Oracle Java ME SDK Developer's Guide* .

Creating and Running Java ME Application in NetBeans

This section describes how to create and run a sample application. This sample application obtains an object representing GPIO pin 2 from the `DeviceManager` object, and tries to obtain its high/low value.

In the NetBeans Projects window, you see the sample project with the file `ME8EmbeddedApplication1.java`. Follow these steps:

1. Double-click the `ME8EmbeddedApplication1.java` file in the Projects window.

2. Copy the following sample code and paste it in the Source window.

```
package me8embeddedapplication1;

import jdk.dio.DeviceManager;
import jdk.dio.UnavailableDeviceException;
import jdk.dio.gpio.GPIOPin;
import java.io.IOException;
```

```
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.microedition.midlet.*;


public class ME8EmbeddedApplication1 extends MIDlet {

    @Override
    public void startApp() {

        try {
            GPIOPin pin = (GPIOPin)DeviceManager.open(2);
            boolean b = pin.getValue();
        } catch (UnavailableDeviceException ex) {
        } catch (IOException ex) {
            Logger.getLogger(ME8EmbeddedApplication1.class.getName()).log(
Level.SEVERE, null, ex);
        }

    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }
}
```

3. Clean and build the `ME8EmbeddedApplication1` project by clicking on the hammer-and-broom icon in the NetBeans toolbar or by selecting **Run** then **Clean and Build Project (ME8EmbeddedApplication1)**.

4. Run the newly cleaned and built `ME8EmbeddedApplication1` project by selecting the green right-arrow icon in the NetBeans toolbar or by selecting **Run** then **Run Project (ME8EmbeddedApplication1)**.

   When the run is successful, the EmbeddedExternalDevice1 emulator starts with the `ME8EmbeddedApplication1` suite running.

Debugging an IMlet on the Raspberry Pi Board

Follow these steps to debug an IMlet using NetBeans:

1. Open the source code of your IMlet class on the NetBeans editor.

2. Click once directly on the line number where you want to set a breakpoint. The line number is replaced by a red square and the line is highlighted in red.

3. Select **Debug** then **Debug Project** or use the **Debug** button on the toolbar.

The debugger connects to the debug agent on the board and the program stops at your breakpoint.

Access to Peripherals

Applications that require access to Device I/O APIs must request appropriate permissions in JAD files. For more information about using the Device I/O APIs, see the *Device I/O API 1.1* specification at:

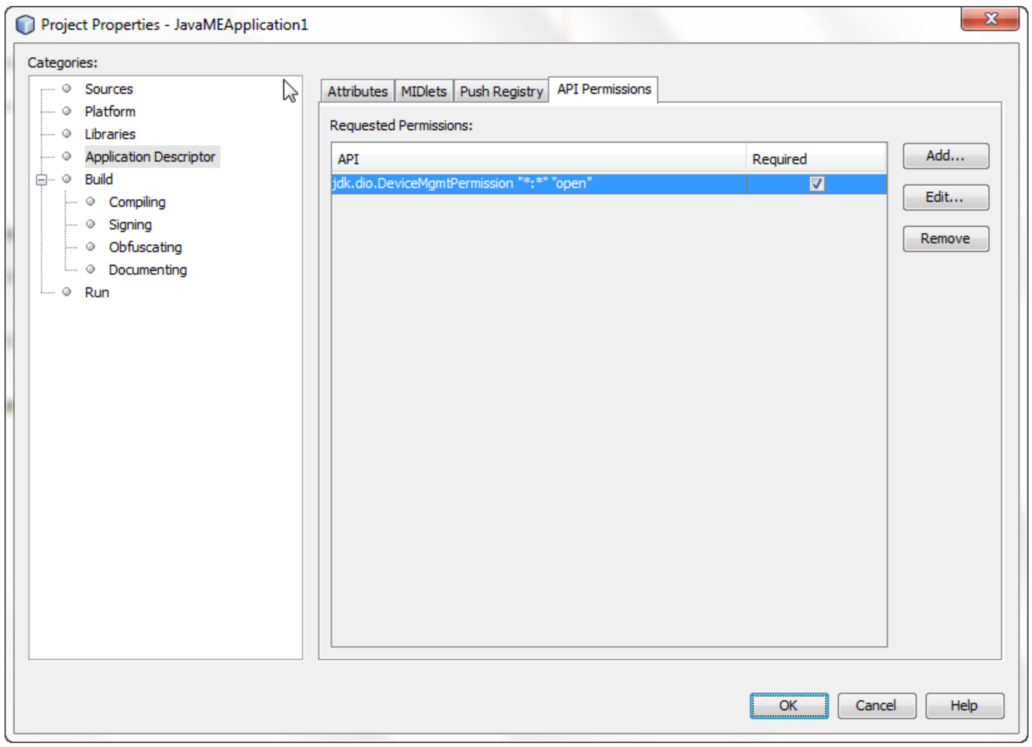
`http://docs.oracle.com/javame/8.2/api/dio/api/index.html`

Signing the Application with API Permissions

The JAD file must have the proper API permissions. Follow these steps to sign the application both in NetBeans and without an IDE:

1. In **NetBeans**, right-click the project name (**ME8EmbeddedApplication1** in this example) and select **Properties**.
2. Click **Application Descriptor**, then in the resulting pane, click **API Permissions**.
3. Click the **Add** button, and add the `jdk.dio.DeviceMgmtPermission` API, as shown in Figure 2-6.
4. Click **OK** to close the project properties dialog.

Figure 2-6 Adding API Permissions with NetBeans



5. If you are not using an IDE, then manually modify the application descriptor file to contain the following permissions:

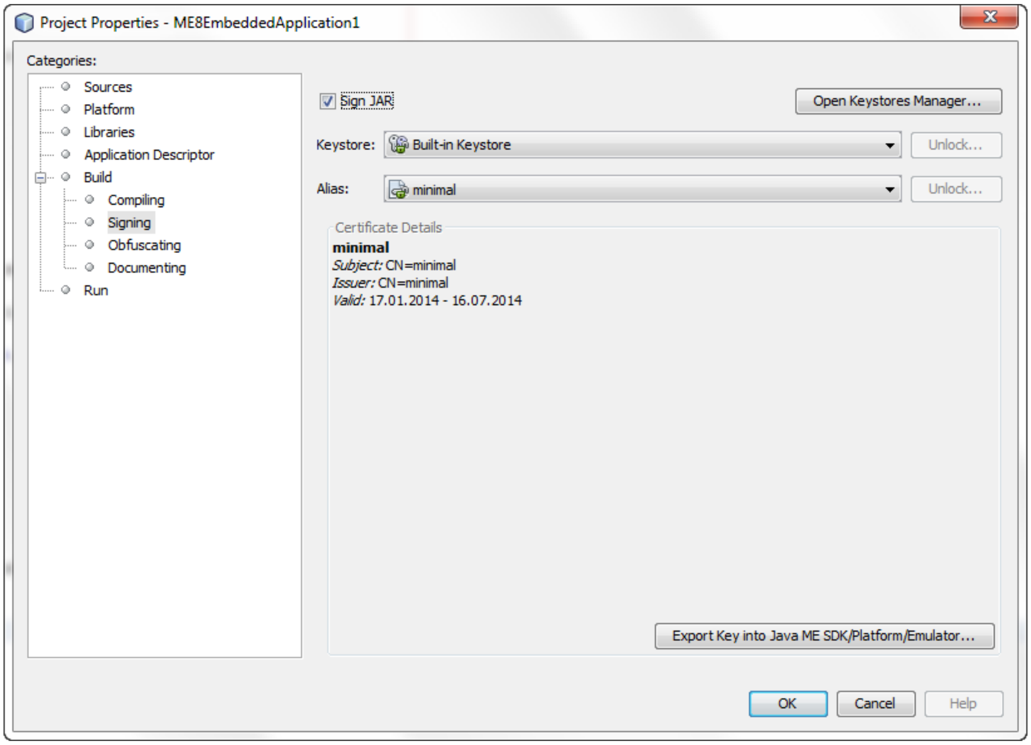
```
MIDlet-Permission-1: jdk.dio.DeviceMgmtPermission "*:*" "open"
```

Method #1: Signing Application Using the NetBeans IDE

The NetBeans IDE enables developers both to sign the applications with a local certificate and upload the certificate on the device. Follow these steps:

1. Right-click the project name and select **Properties**.
2. Under the **Build** category, click **Signing**.
3. Select **Sign JAR** and specify a certificate to sign with as shown in Figure 2-7.

Figure 2-7 Signing Application JAR with NetBeans



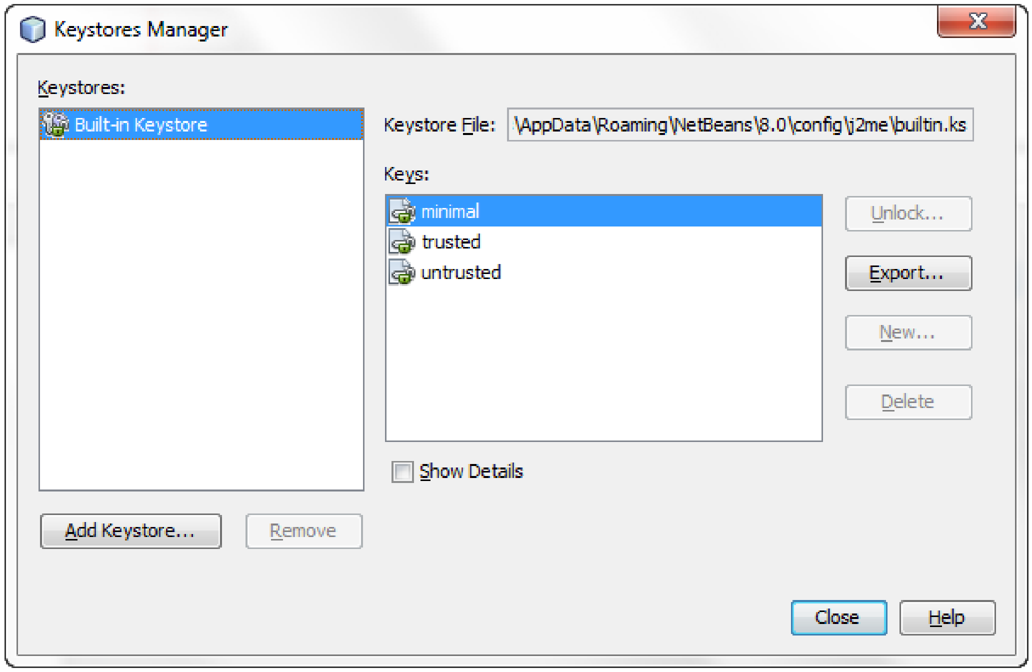
Note:

The selected certificate must be uploaded on the device and associated with the security client.

---

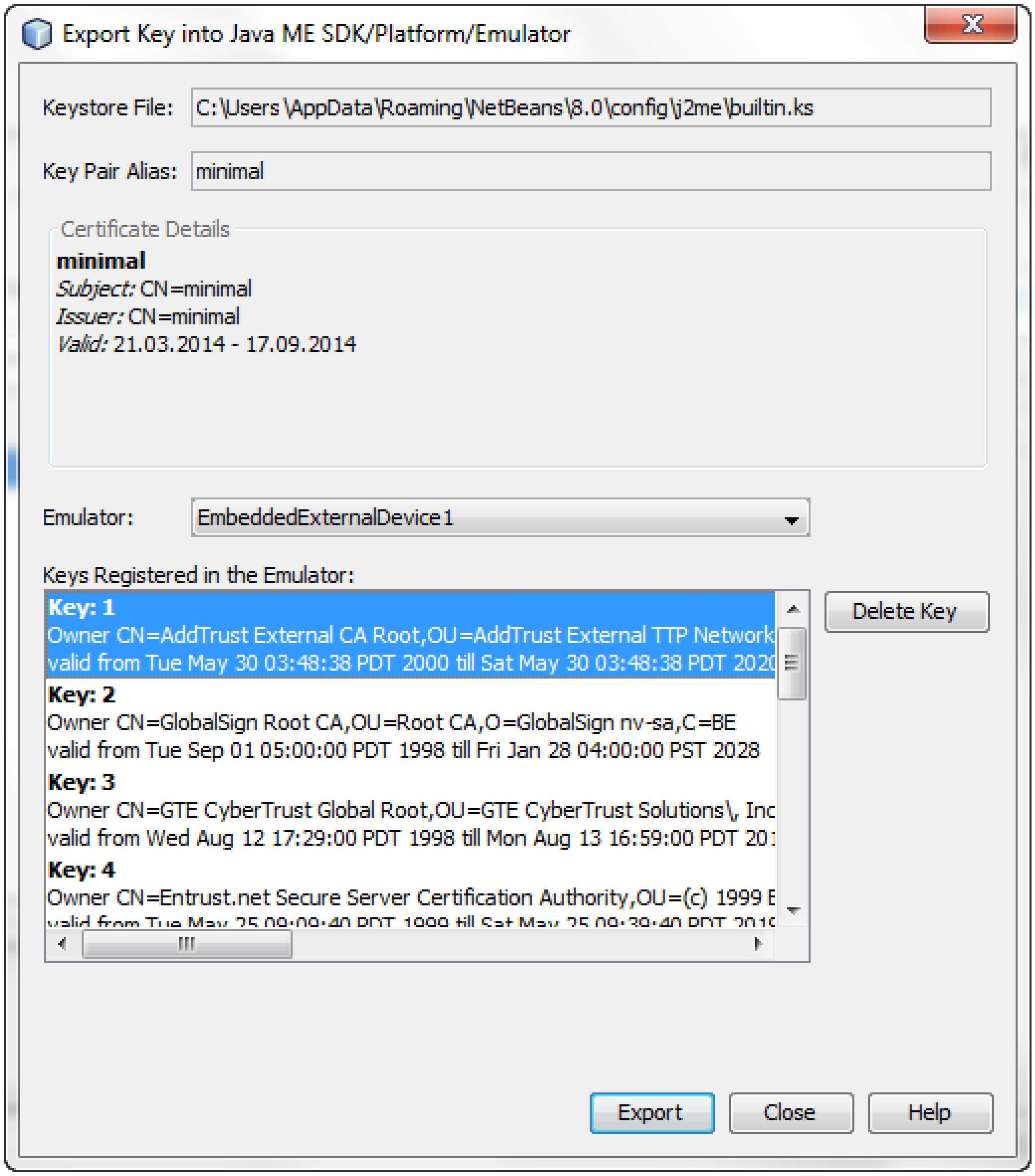
4. Click **Open Keystores Manager**.
5. Select the key and click **Export** as shown in Figure 2-8.

Figure 2-8 Keystores Manager Window



6. In the Export Key window, select the **EmbeddedExternalDevice1** in the Emulator drop-down list, select the certificate in the Security Domain drop-down list, and click **Export** as shown in Figure 2-9.

Figure 2-9 Exporting Key on a Device



7. Download the `_policy.txt` file from the `/appdb` directory of the Raspberry Pi board and add a section with the client name and a set of permissions. For more information about the policy file format, see the External Client Policy Format section in the *Java ME Embedded Profile 8* specification.

8. Ensure that the certificate with the specified common name (CN) is associated with the client by adding a section similar to the following one.

```
client Signed [C=US,O=manufacturer CA,OU=TCK,CN=thehost]
```

9. Copy the modified `_policy.txt` file back to the `/appdb` directory on the Raspberry Pi board.

Method #2: Signing Application Using a Command Line

If you are not using the NetBeans IDE, then you can sign your application using the command line. Follow the instructions on how to set up a keystore with a local certificate that can be used to sign the applications:

1. Generate a new self-signed certificate with the following command on the desktop, using the `keytool` that is shipped with the Oracle Java SE JDK.

   ```
   keytool -genkey -v -alias mycert -keystore mykeystore.ks -
   storepass spass -keypass kpass -validity 360 -keyalg rsa -
   keysize 2048 -dname "CN=thehost"
   ```

   This command generates a 2048-bit RSA key pair and a self-signed certificate, placing them in a new keystore with a keystore password of `spass` and a key password of `kpass` that is valid for 360 days. You can change both passwords as desired.

2. Copy the `certs` directory from the Raspberry Pi board over to the desktop using an `sftp` client or `scp` command, change into the `certs` directory, and perform the following command using the `mekeytool.exe` command (or alternatively `java -jar MEKeyTool.jar...` if your distribution contains only that) that ships with the Oracle Java ME SDK 8.3 distribution.

   ```
   {mekeytool} -import -MEkeystore _main.ks -keystore
   mykeystore.ks -storepass spass -alias mycert -domain trusted
   ```

   This command imports the information in `mykeystore.ks` that you just created to the `_main.ks` keystore. After this is completed, copy the `certs` directory back to the Raspberry Pi board by using an `sftp` client or `scp` command.

3. Use the following commands to sign your application before deploying it to the Raspberry Pi board:

   ```
   jadtool -addcert -chainnum 1 -alias myalias -keystore mykeystore.ks
   -storepass spass -inputkad myjad.jad -outputjad myjad.jad
   ```

   To sign with the SHA256 signature algorithm, add the `-useSha256` parameter. If not present, the default algorithm SHA1withRSA is used.

   ```
   jadtool -addjarsig -chainnum 1 -jarfile myjar.jar -alias myalias -
   keystore mykeystore.ks -storepass spass -keypass kpass -inputjad
   myjad.jad -outputjad myjad.jad -useSha256
   ```

Method #3: Using NullAuthenticationProvider

This method allows you to bypass a certificate check and run unsigned applications as if they were signed and given all requested permissions. Use this method only for development and debugging. Perform final testing using a real certificate as described in method #1 or #2.

1. Stop the Java runtime.

2. To use `NullAuthenticationProvider`, set the following property in the `jwc_properties.ini` file on the Raspberry Pi board:

   ```
   [internal]
   authentication.provider = com.oracle.meep.security.NullAuthenticationProvider
   ```

3. Restart the Java runtime.

Activating Stronger Signature Verification

To activate a stronger signature verification (SHA256) on your device, edit the `jwc_properties.ini` file as follows:

```
ams.verifier.requires_sha256 = true
```

Obtaining Java Logs from a Device

There are two ways in Oracle Java ME 8.3 to obtain a device log:

- Obtaining Java Logs Via an SDK Output Console Window
- Obtaining Java Logs Via a Console Application

These options are mutually exclusive because each of them requires the Developer Agent program but only one instance of the Developer Agent program can be run.

Obtaining Java Logs Via an SDK Output Console Window

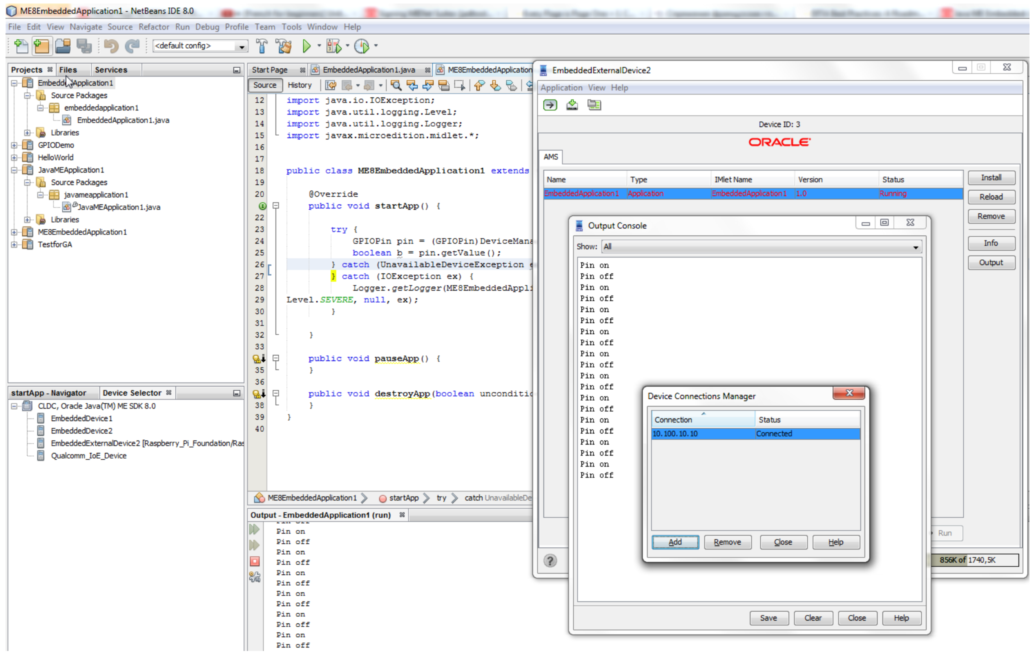
You can obtain Java Logs via an SDK Output Console window with or without using the NetBeans IDE.

If you want to use the NetBeans IDE, follow these steps:

1. Run the NetBeans IDE.
2. Select an IMlet in the Projects window and run it.

   The NetBeans IDE opens the EmbeddedExternalDevice1 window.
3. Click the **Output** button. The log is available in the SDK EmbeddedExternalDevice1 Output Console window shown in Figure 2-10.

Figure 2-10 Java Logging Through the SDK Output Console Using NetBeans IDE



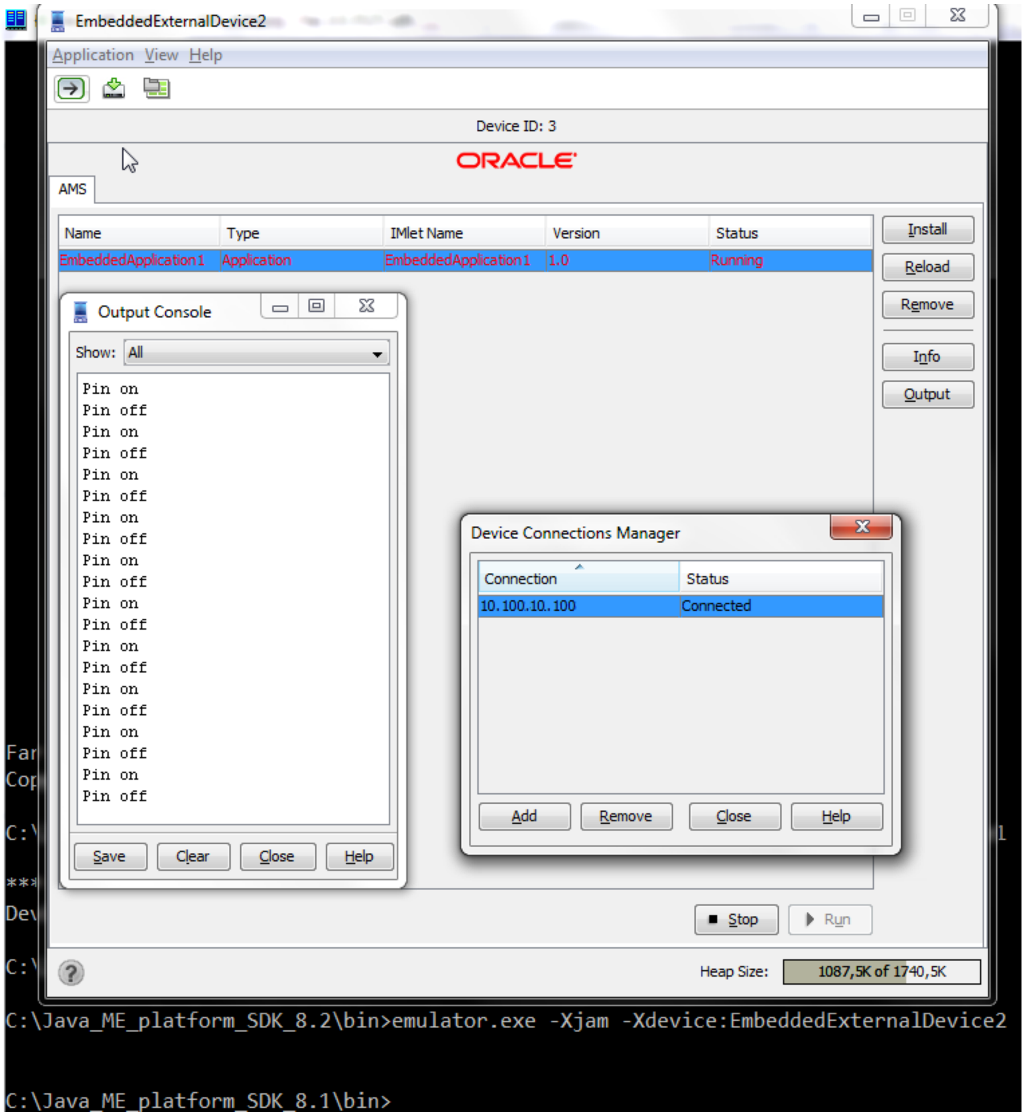
If you want to obtain Java Logs without using the NetBeans IDE, follow these steps:

1. Run the SDK Device Connections Manager located at *<SDK Installation Folder>*`/bin/device-manager.exe`.
2. Wait until the device connection status displays **Connected**.
3. Run the SDK EmbeddedExternalDevice1 by using the following command:

```
emulator.exe -Xjam -Xdevice:EmbeddedExternalDevice1
```

4. Install and run an IMlet using the GUI of the SDK EmbeddedExternalDevice1 window.
5. Click the **Output** button. The log is available in the SDK EmbeddedExternalDevice1 Output Console window shown in Figure 2-11.

Figure 2-11 Java Logging Using the SDK Output Console



Obtaining Java Logs Via a Console Application

To obtain Java Logs using a console application such as Windows Command Line or Far, you must run the Developer Agent program manually. Follow these steps:

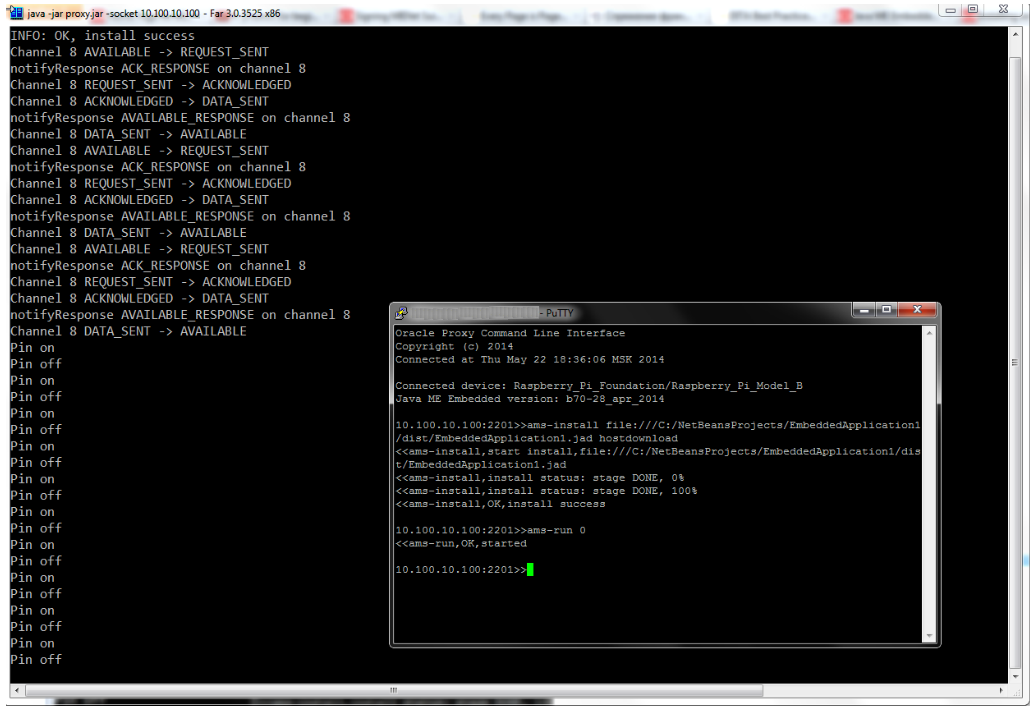
1. Start a console application and enter the following command specifying the IP address of the Raspberry Pi board:

   ```
   java -jar proxy.jar -socket <IP Address>
   ```

2. Install and run IMlets using the CLI.

3. The log will be available in the same console window shown in Figure 2-12.

Figure 2-12 Java Logging Using a Console Application



List of Logging Parameters in the jwc_properties.ini File

You can control the scope of information being logged by editing the following parameters in the `jwc_properties.ini` file:

Table 2-2 Logging Parameters

Parameter	Functional Area
`LC_STORAGE`	The information storage subsystem. The generated messages are useful when facing issues with installing or running applications.
`LC_RMS`	The Record Management System.
`LC_SECURITY`	The security provisioning system.
`LC_PROTOCOL`	GCF
`LC_AMS`	The generated messages are useful when facing issues with installing or running applications.

To select a level of logging details, set a parameter value as follows.

- 0 - all information messages
- 1 - warning level messages
- 2 - errors
- 3 - critical (system's functioning is not guaranteed)
- 4 - logging is turned off

Work With an Embedded Display

This section provides helpful information for working with an embedded display. The section contains the following topics:

- Setting Up a Primary Display
- The Embedded Display ID

Setting Up a Primary Display

If two displays are connected to your board, you can set the primary display by editing the `frame_buffer.device` property in the `jwc_properties.ini` file. For more information about editing the `jwc_properties.ini` file, see Configuring the Java Runtime Properties.

The Embedded Display ID

The name of the embedded display is available in Linux in the `/dev` directory and has the form `/dev/fb*`.

To access the embedded display in your applications, use the `Display.getDisplays()` method. The ID of the embedded display corresponds to the name of the device in Linux.

3

Troubleshooting

This chapter contains a list of common problems that you may encounter while installing and running the Oracle Java ME SDK and embedded software on the Raspberry Pi board. This chapter provides information on the causes of these problems and possible solutions for them.

The common problems in this chapter are grouped in the following categories:

- Installing Linux on the Raspberry Pi Board
- Starting Oracle Java ME Embedded Software on the Board
- Using the Board with the Oracle Java ME SDK and the NetBeans IDE

Installing Linux on the Raspberry Pi Board

Table 3-1 contains information about problems and solutions when installing Linux on the board.

Table 3-1 Problems and Solutions: Installing Linux on the Board

Problem	Cause	Solution
Red power LED is blinking.	The power supply is malfunctioning.	Replace the power supply.
Red power LED is on, but there is no activity from the green LED.	The Raspberry Pi board cannot find a valid disk image on the SD card.	Use a special disk image utility to write the Wheezy disk image onto the SD card. Do not copy the IMG file onto the SD card and attempt to use that to power up the board.
Green LED blinks with a specific pattern	A file needed by the Raspberry Pi board is missing or corrupted.	Replace the following files: • 3 flashes: `loader.bin` not found • 4 flashes: `loader.bin` not started • 5 flashes: `start.elf` not found • 6 flashes:`start.elf` not started • 7 flashes: `kernel.img` not found

Starting Oracle Java ME Embedded Software on the Board

Table 3-2 contains information about problems and solutions when starting the runtime on the board.

Table 3-2 Problems and Solutions: Starting Oracle Java ME Embedded Software on the Board

Problem	Cause	Solution
Oracle Java ME Embedded applications do not start.	The permissions on the distribution files are not set correctly.	Reset the permissions on all files in the distribution to 777.
Oracle Java ME Embedded fails to initialize the network on the board.	The network configuration is incorrect.	Verify that the network connection is correct. Ensure that the board is using DHCP to obtain an IP address.
The Raspberry Pi desktop does not start after booting.	The board does not have the startup sequence activated.	Use the Raspberry Pi setup application to set the desktop to activate at startup.

Using the Board with the Oracle Java ME SDK and the NetBeans IDE

Table 3-3 contains information about problems and solutions when using the board with the Oracle Java ME SDK and the NetBeans IDE.

Table 3-3 Problems and Solutions: Oracle Java ME SDK and the NetBeans IDE

Problem	Cause	Solution
The board is not detected when adding a new device to the Device Selector.	LAN proxy settings of the host might be a source of the problem.	The socket proxy needs to be disabled: Open the Proxy Settings window available by clicking the **Advanced** button on the LAN Setting window. Ensure that the **Socks** field is empty and the **Use the same proxy server for all protocols** check box is not selected.
	The proxy connection is not enabled or an improper proxy host address is set up in the `jwc_properties.ini` file.	Ensure that the `+UseProxy` parameter is present in the `run.sh` script. Also, if the Developer Agent program runs in a client mode, ensure that the `proxy.client_connection_address` property is properly configured to use the Developer Agent program's host address.
The debugging session freezes, disconnects unexpectedly, or shows error messages.	The firewall on the computer is blocking some debugging traffic.	Open TCP port 2808 on your firewall configuration settings. The exact procedure to open a port differs depending on your version of Windows or your firewall software.
	Thunderbird is using a port that is needed for communication with the board.	Close `thunderbird.exe` during the debugging session.
Installation of a big MIDlet results in the error "AMS generated out of memory".	There is not enough memory for AMS operation.	It is recommended that you stop other applications that consume memory resources and then proceed with the installation of a big MIDlet.

Table 3-3 (Cont.) Problems and Solutions: Oracle Java ME SDK and the NetBeans IDE

Problem	Cause	Solution
The board IP address is not shown by default in the Java ME SDK Device Manager "Add Device" dialog. The SDK device-manager.log is full with the following error messages: UDP device detection failed java.net.BindException: Address already in use	The ports used for IP-based device detection are busy on the host machine. By default, they are UDP IPv4 ports 55208 and 55209.	Perform one of the following actions: • Terminate an application which uses these ports. • Configure the board and Java ME SDK to use different ports. Ports are set on the device by the properties proxy.udp_device_detection_request_port and proxy.udp_device_detection_response_port. To modify the values of these properties, create the proxyOptions.txt file and save it in the <*SDK installation folder*>/toolkit-lib/lib folder. Add the following line: -bcastports <request> <response> where request and response are new values of the properties proxy.udp_device_detection_request_port and proxy.udp_device_detection_response_port.

A

Device I/O Preconfigured List

This appendix describes the proper ID and names for the various peripheral ports and buses for the Raspberry Pi embedded board, which are accessible using the Device I/O APIs.

Note that any IMlet that accesses the Device I/O APIs must be digitally signed using a trusted certificate authority. An IMlet that is not signed will encounter an authentication error when attempting to access the Device I/O APIs.

To access any device from the preconfigured peripheral list, the following permission is required:

```
jdk.dio.DeviceMgmtPermission(%Name%:%ID%);
```

You can find the names and IDs for specific devices in the tables that follow in this appendix. You must also specify an action. An empty string means `open`.

The tables use the following legend:

- **Device ID**: an integer identifier that can be used to open the device with the methods of the `DeviceManager` class.
- **Device Name**: the string name of a device that can be used to open it by name with the methods of the `DeviceManager class`.
- **Mapped**: all hardware-related information regarding a peripheral, such as physical location, mapping, or port. This information enables the user to determine the peripheral's location on a target board.
- **Configuration**: properties that are passed to the specific `DeviceConfig` constructor to open the peripheral by ID or name. The configuration can be used to open the peripheral using the `DeviceManager` with the appropriate configuration.

Note the following items for Device I/O in the Raspberry Pi board:

- The interface `DeviceConfig.HardwareAddressing` supports device names only with UART devices. Do not use the `DeviceConfig.HardwareAddressing.getDeviceName()` method.
- The `PulseCounter` instance cannot be opened by the `PulseCounterConfig` instance with the `GPIOPinConfig` instance specified.
- The `PWMChannel` instance cannot be opened by the `PWMChannelConfig` class with the `GPIOPinConfig` instance specified.

GPIO Pins

The following GPIO pins are preconfigured.

Devicel ID	Device Name	Mapped	Configuration

1	GPIO4	GPIO 4	`controllerNumber = 0`
			`pinNumber = 4`
			`direction = GPIOPinConfig.DIR_INPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES`
			`initValue` - ignored

2	GPIO7	GPIO 7	`controllerNumber = 0`
			`pinNumber = 7`
			`direction = GPIOPinConfig.DIR_OUTPUT_ONLY`
			`mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL`
			`trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES`
			`initValue = false`

3	GPIO17	GPIO 17	`controllerNumber = 0`
			`pinNumber = 17`
			`direction = GPIOPinConfig.DIR_INPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES`
			`initValue` - ignored

4	GPIO18	GPIO 18	`controllerNumber = 0`
			`pinNumber = 18`
			`direction = GPIOPinConfig.DIR_OUTPUT_ONLY`
			`mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL`
			`trigger` - ignored
			`initValue = false`

Devicel ID	Device Name	Mapped	Configuration

5	GPIO22	GPIO 22	`controllerNumber = 0`
			`pinNumber = 22`
			`direction = GPIOPinConfig.DIR_INPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES`
			`initValue` - ignored

6	GPIO23	GPIO 23	`controllerNumber = 0`
			`pinNumber = 23`
			`direction = GPIOPinConfig.DIR_OUTPUT_ONLY`
			`mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL`
			`trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES`
			`initValue = false`

7	GPIO24	GPIO 24	`controllerNumber = 0`
			`pinNumber = 24`
			`direction = GPIOPinConfig.DIR_OUTPUT_ONLY`
			`mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL`
			`trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES`
			`initValue = false`

8	GPIO25	GPIO 25	`controllerNumber = 0`
			`pinNumber = 25`
			`direction = GPIOPinConfig.DIR_OUTPUT_ONLY`
			`mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL`
			`trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES`
			`initValue = false`

Devicel ID	Device Name	Mapped	Configuration

9	GPIO27	GPIO 27	`controllerNumber = 0`
			`pinNumber = 27`
			`direction = GPIOPinConfig.DIR_INPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES`
			`initValue` - ignored

Please note the following items concerning GPIO on the Raspberry Pi board.

- The value of `DeviceConfig.DEFAULT` when applied to the `controllerNumber` is 0.
- The value of `DeviceConfig.DEFAULT` when applied to the `mode` means that the GPIO pin be configured in the default mode, as per the table above.
- GPIO modes are not software-configurable. All GPIO pins in the preceding table are given with the only mode that is supported on the Raspberry Pi. If an application attempts to configure a GPIO pin to use an unsupportable mode, an exception will be thrown.
- For GPIO pins that are configured as input pins, the `initValue` parameter is ignored.
- The trigger modes `TRIGGER_HIGH_LEVEL`, `TRIGGER_LOW_LEVEL`, and `TRIGGER_BOTH_LEVELS` are not supported on the Raspberry Pi.
- For all GPIO pins, the application should pass in a 0 for the GPIO port when necessary.
- The following diagram represents the pin positions of the Raspberry Pi, Revision 1 and 2.

I2C

There is no static I2C configuration with the Raspberry Pi because there is no connected hardware. In comparison with SPI, I2C does not allow any communication with a loopback device.

Device ID	Device Name	Mapped	Configuration
NONE		GPIO 2 (SDA) GPIO 3 (SCL)	

Please note the following items about I2C on the Raspberry Pi.

- `I2CDevicePermission` is necessary.
- For revision 1 boards, I2C is provided by default on GPIO 0 and 1 (bus 0), and for revision 2 boards, I2C is provided on GPIO 2 and 3 (bus 1.)

- The value of `DeviceConfig.DEFAULT` when applied to the `busNumber` is 0.
- The value of `DeviceConfig.DEFAULT` when applied to the `addressSize` is 7.
- The only supported value for the `clockFrequency` field is `DeviceConfig.UNASSIGNED`.
- Before using I2C, you will have to load two I2C modules: `i2c-bcm2708` and `i2c-dev`. Add the following two lines to the `/etc/modules` file and reboot to apply the changes.

```
i2c-bcm2708
i2c-dev
```

MMIO

The following MMIO peripherals are available:

Device ID	Device Name	Mapped	Configuration
900	PWM		

The MMIO peripherals include CTL, STA, RNG1, DAT1, and FIF1 registers (all of them are of type INT) with no event support.

Note: MMIO devices are only available if Java ME runtime runs with superuser privileges as shown in the following example.

```
pi@raspberry ~ /pi/bin $ sudo ./usertest.sh
```

Due to nature of memory organization of the Raspberry Pi, programmers can create a custom `MMIODeviceConfig` to access the memory range {0x20000000, 0x21000000}. Please note that not all addresses are accessible in the range and some of them may cause a board reboot. Please check the documentation for SFR addresses and its behavior. The end addresses are not inclusive.

SPI

The SPI has a single static configuration with the following parameters:

Device ID	Device Name	Mapped	Configuration
300	SPI_Slave	GPIO10 (MOSI) GPIO9 (MISO) GPIO11 (SCLK) GPIO8 (CE0)	SPI bus number: 0 (SPI1) Chip Enable: 0 (CE0/GPIO8) The number of bit of slave's word: 8 Clock frequency in Hz: 2000000 Clock polarity and phase: 1 (CPOL_Low, CPHA_2Edge) Bit ordering of the slave device: 1 (BIG_ENDIAN)

Please note the following items about SPI on the Raspberry Pi.

- The value of `DeviceConfig.DEFAULT` when applied to the `busNumber` is 0.
- The value of `DeviceConfig.DEFAULT` when applied to the `clockFrequency` is 2000000 Hz.
- The value of `DeviceConfig.DEFAULT` when applied to the `wordLength` is 8.
- The value of `DeviceConfig.DEFAULT` when applied to the `bitOrdering` is 1 (big-endian).
- Before using SPI, you will have to load the SPI modules by running the following command: `$ modprobe spi_bcm2708`, or by using the same method as I2C: uncomment the appropriate line in the `etc/modprob.d/raspi-blacklist.conf` file and reboot the board.
- Only 8-bit word lengths are supported on the Raspberry Pi board.
- No real hardware is connected by default.

> **Note:**
>
> You can connect MISO and MOSI pins to get a simple loopback device for testing your code.

UART

The following UART devices are preconfigured:

Device ID	Device Name	Mapped	Configuration

100	UART	GPIO 14 (TXD)
		GPIO 15 (RXD)
			`controllerName = ttyAMA0`
			`baudRate = 19200`
			`dataBits = DATABITS_8`
			`parity = PARITY_NONE`
			`stopBits = STOPBITS_1`
			`flowcontrol = FLOWCONTROL_NONE`
			`inputBufferSize` - ignored
			`outputBufferSize` - ignored

Please note the following items about UART on the Raspberry Pi.

- By default, the Raspberry Pi uses the UART as a serial console. Before using UART, make sure that `/dev/ttyAMA0` is not being used as a console. This can be done by changing the boot command line by editing the `/boot/cmdline.txt` file and removing the line "`console=ttyAMA0,115200 kgdboc=ttyAMA0,115200`" from the boot arguments. Also, comment out the following line: "`2:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100`" in the file `/etc/inittab`.

  In the case when the `/boot` is mounted as a read-only partition, it must be remounted with write permissions to enable modifying the `cmdline.txt` file.

  ```
  sudo mount -n -o remount,rw /boot
  ```

  Then edit the `cmdline.txt` file and reboot the Raspberry Pi board.

- By default, the `pi` user is in the `dialout` group. That gives `pi` the ability to access `/dev/ttyAMA0` (and, consequently, UART from Java) without administrator rights.

Watchdog

The following watchdog devices are pre-configured:

Device ID	Device Name	Mapped	Configuration
1100	WDG	Platform Watchdog	
1101	WWDG		

ATCmdDevice

The following ATCmd device is preconfigured:

Device ID	Device Name	Mapped	Configuration
800	EMUL		A simple ATCmd device emulator. Returns `OK` for every command request. No async notification is supported. For debug purpose only.

Structure of the Device I/O Configuration File

Device I/O configuration file structure is based on the JSON data interchange format with a few distinctions.

General File Data Format

The device I/O configuration file resides in the `/appdb` directory and has the name `daapi_config.json`. Based on the JSON data interchange format, the Device I/O configuration file has the following distinctions:

- A string is quoted only if it contains a space.
- A value can be only a string, object, or array.
- UTF–8 is the only supported encoding.

For more information about the JSON data interchange format, see www.json.org.

Configuration Hierarchy

The configuration fie has three sections.

1. An unnamed global object that contains two other sections. It begins with { (left brace) and ends with } (right brace).
2. `configs`: a container for all configuration objects
3. `defaults`: a container for default properties grouped by peripheral packages

All the sections are mandatory even if there is no content as shown in the following example.

```
{
configs: {}
defaults: {}
}
```

The configs Section

The `configs` section contains a collection of key/value pairs named properties, where the key is an ID assigned to a configuration (see *Device I/O 1.1 API* specification) and the value is an object that contains information specific for the described configuration. The configuration must contain at least the `deviceType` property that points to the peripheral class to which this configuration applies. The value of the `deviceType` property is a shortened form of the peripheral class name. The rest of the properties are device specific. Property names must conform with the *Device I/O 1.1 API* specification. The `name` field can be an array of names to support name aliases.

Example A-1 Example of GPIOPin Device Configuration

An example below shows a valid configuration description. Note that it is mandatory that the `deviceType` and `name` field values are set.

```
10 : {
deviceType : gpio.GPIOPin,
id:1,
deviceNumber : 1,
pinNumber : 5,
direction : 1,
mode:4,
name :[LED_1,GPIO6.6,GPIO0]
initValue:0,
}
```

The defaults Section

The `defaults` section provides default values for some properties and also additional information required for the construction of a peripheral instance. The main purpose of this section is to reduce the amount of information provided by the `configs` section and thus to decrease the file size. The same requirement for key naming applies to the . The properties are grouped by the shortened name of peripheral class for which they may be used. For example, `uart.UARTConfig` is a group for the `com.oracle.deviceaccess.uart.UARTConfig` property object.

Example A-2 Example of the daapi_config.json File

Refer to the following `daapi_config.json` file example.

```
{
configs: {
    13: {
        deviceType: atcmd.ATDevice,
        name: EMUL,
        properties : ["com.oracle.sms=true"]
        },
    15: {
        deviceType: uart.ModemUART,
        deviceNumber: 1,
        name: COM1,
        baudRate: 9600,
```

```
        dataBits: 7,
        },
},
defaults: {
    uart.UARTConfig: {
    baudRate:115200,
    parity:0,
    dataBits:8,
    stopBits:1,
    flowControl:0,
},
}
```

B

Configuring the Java Runtime Properties

This appendix describes how to change the value of a property that affects Java's configuration or behavior at runtime.

The appendix contains the following sections:

- Editing the jwc_properties.ini File
- Using the CLI set-property Command

Editing the jwc_properties.ini File

The `jwc_properties.ini` file contains all the properties that affect Java configuration and behavior at runtime. In order to edit this file, do the following:

1. Stop the Java runtime on the Raspberry Pi board.
2. Open the `jwc_properties.ini` that is a part of the Oracle Java ME Embedded bundle (or download it from the board), find the property that should be changed, and modify its value.
3. Copy the modified version of the `jwc_properties.ini` file to the `/bin` directory on the Raspberry Pi board using the `sftp` client program.
4. Restart Java on the Raspberry Pi board.

Using the CLI set-property Command

To modify a property using the `set-property` command in the command-line interface (CLI), do the following.

1. Connect to the board using command-line interface (CLI).
2. Execute the "`set-property` *<property_name>* *<desired_property_value>*" command.
3. Restart Java on the board.

Note, that by executing the `set-property` command, the `jwc_properties.ini` file is always updated automatically.

Glossary

access point

A network-connectivity configuration that is predefined on a device. An access point can represent different network profiles for the same bearer type, or for different bearer types that may be available on a device, such as WiFi or Bluetooth.

ADC

analog-to-digital converter. A hardware device that converts analog signals (time and amplitude) into a stream of binary numbers that can be processed by a digital device.

AMS

Application Management System. The system functionality that completes tasks such as installing applications, updating applications, and managing applications between foreground and background.

APDU

Application Protocol Data Unit. A communication mechanism used by SIM cards and smart cards to communicate with card reader software or a card reader device.

API

application programming interface. A set of classes used by programmers to write applications that provide standard methods and interfaces and eliminate the need for programmers to reinvent commonly used code.

ARM

Advanced RISC Machine. A family of computer processors using reduced instruction set (RISC) CPU technology, developed by ARM Holdings. ARM is a licensable instruction set architecture (ISA) used in the majority of embedded platforms.

AT commands

A set of commands developed to facilitate modem communications, such as dialing, hanging up, and changing the parameters of a connection. Also known as the Hayes command set. AT means *attention.*

AXF

ARM Executable Format. An ARM executable image generated by ARM tools.

BIP

Bearer Independent Protocol. Allows an application on a SIM card to establish a data channel with a terminal, and through the terminal, to a remote server on the network.

CDMA

Code Division Multiple Access. A mobile telephone network standard used primarily in the United States and Canada as an alternative to GSM.

CLDC

Connected Limited Device Configuration. A Java ME platform configuration for devices with limited memory and network connectivity. It uses a low-footprint Java Virtual Machine such as the CLDC HotSpot Implementation, and several minimalist Java platform APIs for application services.

configuration

Defines the minimum Java runtime environment (for example, the combination of a Java Virtual Machine and a core set of Java platform APIs) for a family of Java ME platform devices.

DAC

digital-to-analog converter. A hardware device that converts a stream of binary numbers into an analog signal (time and amplitude), such as audio playback.

ETSI

European Telecommunications Standards Institute. An independent, non-profit group responsible for the standardization of information and communication technologies within Europe. Although based in Europe, it carries worldwide influence in the telecommunications industry.

GCF

Generic Connection Framework. A Java ME API consisting of a hierarchy of interfaces and classes to create connections (such as HTTP, datagram, or streams) and perform I/O.

GPIO

general purpose I/O. Unassigned pins on an embedded platform that can be assigned or configured as needed by a developer.

GPIO port

A group of GPIO pins (typically 8 pins) arranged in a group and treated as a single port.

GSM

Global System for Mobile Communications. A 3G mobile telephone network standard used widely in Europe, Asia, and other parts of the world.

HTTP

HyperText Transfer Protocol. The most commonly used Internet protocol, based on TCP/IP that is used to fetch documents and other hypertext objects from remote hosts.

HTTPS

Secure HyperText Transfer Protocol. A protocol for transferring encrypted hypertext data using Secure Sockets Layer (SSL) technology.

I2C

Inter-Integrated Circuit. A multimaster, serial computer bus used to attach low-speed peripherals to an embedded platform

ICCID

Integrated Circuit Card Identification. The unique serial number assigned to an individual SIM card.

IMP-NG

Information Module Profile Next Generation. A profile for embedded "headless" devices, the IMP-NG specification (JSR 228) is a subset of MIDP 2.0 that leverages many of the APIs of MIDP 2.0, including the latest security and networking+, but does not include graphics and user interface APIs.

IMEI

International Mobile Equipment Identifier. A number unique to every mobile phone. It is used by a GSM or UMTS network to identify valid devices and can be used to stop a stolen or blocked phone from accessing the network. It is usually printed inside the battery compartment of the phone.

IMlet

An application written for IMP-NG. An IMlet does not differ from MIDP 2.0 MIDlet, except by the fact that an IMlet cannot refer to MIDP classes that are not part of IMP-NG. An IMlet can only use the APIs defined by the IMP-NG and CLDC specifications.

IMlet Suite

A way of packaging one or more IMlets for easy distribution and use. Similar to a MIDlet suite, but for smaller applications running in an embedded environment.

IMSI

International Mobile Subscriber Identity. A unique number associated with all GSM and UMTS network mobile phone users. It is stored on the SIM card inside a phone and is used to identify itself to the network.

ISA

Instruction Set Architecture. The part of a computer's architecture related to programming, including data type, addressing modes, interrupt and exception handling, I/O, and memory architecture, and native commands. Reduced instruction set computing (RISC) is one kind of instruction set architecture.

JAD file

Java Application Descriptor file. A file provided in a MIDlet or IMlet suite that contains attributes used by application management software (AMS) to manage the MIDlet or IMlet life cycle, and other application-specific attributes used by the MIDlet or IMlet suite itself.

JAR file

Java ARchive file. A platform-independent file format that aggregates many files into one. Multiple applications written in the Java programming language and their required components (class files, images, sounds, and other resource files) can be bundled in a JAR file and provided as part of a MIDlet or IMlet suite.

Java ME platform

Java Platform, Micro Edition. A group of specifications and technologies that pertain to running the Java platform on small devices, such as cell phones, pagers, set-top boxes, and embedded devices. More specifically, the Java ME platform consists of a configuration (such as CLDC) and a profile (such as MIDP or IMP-NG) tailored to a specific class of device.

JCP

Java Community Process. The global standards body guiding the development of the Java programming language.

JSR

Java Specification Request. A proposal for developing new Java platform technology, which is reviewed, developed, and finalized into a formal specification by the JCP program.

JVM

Java Virtual Machine. A software “execution engine" that safely and compatibly executes the byte codes in Java class files on a microprocessor.

KVM

A Java Virtual Machine designed to run in a small, limited-memory device. The CLDC configuration was initially designed to run in a KVM.

LCDUI

Liquid Crystal Display User Interface. A user interface toolkit for interacting with liquid crystal display (LCD) screens in small devices. More generally, a shorthand way of referring to the MIDP user interface APIs.

MIDlet

An application written for MIDP.

MIDlet suite

A way of packaging one or more MIDlets for easy distribution and use. Each MIDlet suite contains a Java Application Descriptor file (`.jad`), which lists the class names and files names for each MIDlet, and a Java ARchive file (`.jar`), which contains the class files and resource files for each MIDlet.

MIDP

Mobile Information Device Profile. A specification for a Java ME platform profile, running on top of a CLDC configuration that provides APIs for application life cycle, user interface, networking, and persistent storage in small devices.

MSISDN

Mobile Station Integrated Services Digital Network. A number uniquely identifying a subscription in a GSM or UMTS mobile network. It is the telephone number to the SIM card in a mobile phone and used for voice, FAX, SMS, and data services.

MVM

Multiple Virtual Machines. A software mode that can run more than one MIDlet or IMlet at a time.

obfuscation

A technique used to complicate code by making it harder to understand when it is decompiled. Obfuscation makes it harder to reverse-engineer applications and therefore, steal them.

optional package

A set of Java ME platform APIs that provides additional functionality by extending the runtime capabilities of an existing configuration and profile.

preverification

Due to limited memory and processing power on small devices, the process of verifying Java technology classes is split into two parts. The first part is preverification which is done off-device using the preverify tool. The second part, which is verification, occurs on the device at runtime.

Profile

A set of APIs added to a configuration to support specific uses of an embedded or mobile device. Along with its underlying configuration, a profile defines a complete and self-contained application environment.

Provisioning

A mechanism for providing services, data, or both to an embedded or mobile device over a network.

Pulse Counter

A hardware or software component that counts electronic pulses, or events, on a digital input line, for example, a GPIO pin.

Push Registry

The list of inbound connections, across which entities can push data. Each item in the list contains the URL (protocol, host, and port) for the connection, the entity permitted to push data through the connection, and the application that receives the connection.

RISC

reduced instruction set computing. A CPU design based on simplified instruction sets that provide higher performance and faster execution of individual instructions. The ARM architecture is based on RISC design principles.

RL-ARM

Real-Time Library. A group of tightly coupled libraries designed to solve the real-time and communication challenges of embedded systems based on ARM processor-based microcontroller devices.

RMI

Remote Method Invocation. A feature of Java SE technology that enables Java technology objects running in one virtual machine to seamlessly invoke objects running in another virtual machine.

RMS

Record Management System. A simple record-oriented database that enables an IMlet or MIDlet to persistently store information and retrieve it later. MIDlets can also use the RMS to share data.

RTOS

Real-Time Operating System. An operating system designed to serve real-time application requests. It uses multi-tasking, an advanced scheduling algorithm, and minimal latency to prioritize and process data.

RTSP

Real Time Streaming Protocol. A network control protocol designed to control streaming media servers and media sessions.

SCWS

Smart Card Web Server. A web server embedded in a smart card (such as a SIM card) that allows HTTP transactions with the card.

SD card

Secure Digital cards. A nonvolatile memory card format for use in portable devices, such as mobile phones and digital cameras, and embedded systems. SD cards come in three different sizes, with several storage capacities and speeds.

SIM

Subscriber Identity Module. An integrated circuit embedded into a removable SIM card that securely stores the International Mobile Subscriber Identity (IMSI) and the related key used to identify and authenticate subscribers on mobile and embedded devices.

Slave mode

Describes the relationship between a master and one or more devices in a Serial Peripheral Interface (SPI) bus arrangement. Data transmission in an SPI bus is initiated by the master device and received by one or more slave devices, which cannot initiate data transmissions on their own.

smart card

A card that stores and processes information through the electronic circuits embedded in silicon in the substrate of its body. Smart cards carry both processing power and information. A SIM card is a special kind of smart card for use in a mobile device.

SMS

Short Message Service. A protocol allowing transmission of short text-based messages over a wireless network. SMS messaging is the most widely-used data application in the world.

SMSC

Short Message Service Center. Routes messages and regulates traffic. When an SMS message is sent, it goes to an SMS center first, and then gets forwarded to the destination. If the destination is unavailable (for example, the recipient embedded board is powered down), the message is stored in the SMSC until the recipient becomes available.

SOAP

Simple Object Access Protocol. An XML-based protocol that enables objects of any type to communicate in a distributed environment. It is most commonly used to develop web services.

SPI

Serial Peripheral Interface. A synchronous bus commonly used in embedded systems that allows full-duplex communication between a master device and one or more slave devices.

SSL

Secure Sockets Layer. A protocol for transmitting data over the Internet using encryption and authentication, including the use of digital certificates and both public and private keys.

SVM

Single Virtual Machine. A software mode that can run only one MIDlet or IMlet at a time.

task

At the platform level, each separate application that runs within a single Java Virtual Machine is called a task. The API used to instantiate each task is a stripped-down version of the Isolate API defined in JSR 121.

TCP/IP

Transmission Control Protocol/Internet Protocol. A fundamental Internet protocol that provides for reliable delivery of streams of data from one host to another.

Terminal Profile

Device characteristics of a terminal (mobile or embedded device) passed to the SIM card along with the IMEI at SIM card initialization. The terminal profile tells the SIM card what values are supported by the device.

UART

Universal Asynchronous Receiver/Transmitter. A piece of computer hardware that translates data between serial and parallel formats. It is used to facilitate communication between different kinds of peripheral devices, input/output streams, and embedded systems, to ensure universal communication between devices.

UICC

Universal Integrated Circuit Card. The smart card used in mobile terminals in GSM and UMTS networks. The UICC ensures the integrity and security of personal data on the card.

UMTS

Universal Mobile Telecommunications System. A third-generation (3G) mobile communications technology. It utilizes the radio spectrum in a fundamentally different way than GSM.

URI

Uniform Resource Identifier. A compact string of characters used to identify or name an abstract or physical resource. A URI can be further classified as a uniform resource locator (URL), a uniform resource name (URN), or both.

USAT

Universal SIM Application Toolkit. A software development kit intended for 3G networks. It enables USIM to initiate actions that can be used for various value-added services, such as those required for banking and other privacy-related applications.

USB

Universal Serial Bus. An industry standard that defines the cables, connectors, and protocols used in a bus for connection, communication, and power supply between computers and electronic devices, such as embedded platforms and mobile phones.

USIM

Universal Subscriber Identity Module. An updated version of a SIM designed for use over 3G networks. USIM is able to process small applications securely using better cryptographic authentication and stronger keys. Larger memory on USIM enables the addition of thousands of details including subscriber information, contact details, and other custom settings.

WAE

Wireless Application Environment. An application framework for small devices, which leverages other technologies, such as Wireless Application Protocol (WAP).

WAP

Wireless Application Protocol. A protocol for transmitting data between a server and a client (such as a cell phone or embedded device) over a wireless network. WAP in the wireless world is analogous to HTTP in the World Wide Web.

watchdog timer

A dedicated piece of hardware or software that "watches" an embedded system for a fault condition by continually polling for a response. If the system goes offline and no response is received, then the watchdog timer initiates a reboot procedure or takes other steps to return the system to a running state.

WCDMA

Wideband Code Division Multiple Access. A detailed protocol that defines how a mobile phone communicates with the tower, how its signals are modulated, how datagrams are structured, and how system interfaces are specified.

WMA

Wireless Messaging API. A set of classes for sending and receiving Short Message Service (SMS) messages.

XML Schema

A set of rules to which an XML document must conform to be considered valid.