Oracle® Java ME Embedded

Getting Started Guide for the Reference Platform (Raspberry Pi)

Release 8

E48512-03

April 2014

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

E48512-03

Contents

List of Figures

List of Tables

List of Examples

Preface

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

Audience

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

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

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 software on the Raspberry Pi board, installing and using the Developer Agent program on the desktop host, configuring the Oracle Java ME Embedded system, connecting to the Raspberry Pi using a secure shell, and installing and running an Oracle Java ME Embedded application.

The following items are required for developing on the Raspberry Pi board:

- Raspberry Pi Rev. B 512 MB board
- Oracle Java ME Embedded 8 distribution
- Micro-USB power supply of .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 and mouse, as well as 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 connection, 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.

Downloading and Installing the PuTTY Terminal Emulator Program

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

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

The terminal emulator executable file is directly downloadable as `putty.exe`. The terminal emulator is used to connect to the AMS command-line interface (CLI) that sends commands to the board.

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.

Preparing the Raspberry Pi Board

To develop applications on the Raspberry Pi board, you must first download and install the Wheezy variant of Raspbian Debian Linux on the Raspbian Pi board. To do this, follow these steps:

1. Download the Raspbian Wheezy hard-float (Debian Linux) 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 will 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. Connect the RJ-45 network cable, monitor, keyboard, and mouse.
6. Install the SD card in the Raspberry Pi board.
7. 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.
8. If the Linux installation was successful, the Raspberry Pi board will start and obtain a DHCP address.
9. 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`.
10. 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.
11. Log in if necessary, and if you are using the desktop, start an LXTerminal.
12. Run the `ifconfig` command to display the Raspberry Pi IP address. This is necessary so you can access and control the board remotely. Remember this IP address; it will be used in the next set of steps.

Installing the Oracle Java ME Embedded Software

Version 8 of the Oracle Java ME Embedded software contains a different architecture than previous versions. With version 8, the user has the option to run a Developer Agent program on the desktop under Windows. Commands that are sent to the board from the Windows 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 Pi and contains internal Java libraries
- `/bin`: This directory is used on the 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.

You must make two copies of the Oracle Java ME Embedded ZIP archive file. The first remains on the Windows desktop, while the second must be transferred to the Raspberry Pi board.

1. Use an `sftp` client or `scp` command to transfer one copy of the Oracle Java ME Embedded ZIP 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]
   ```

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

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

3. 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:

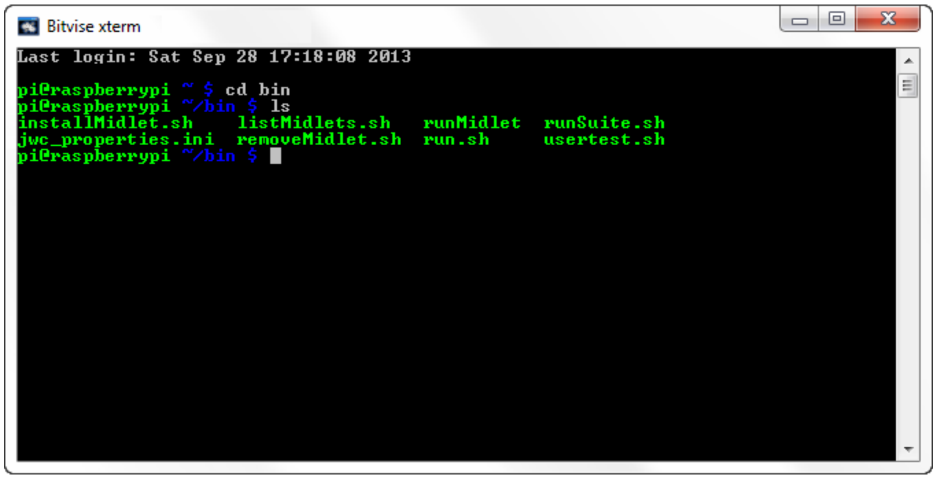
   ```
   $ssh -l pi [IP address of board]
   ```

4. Unzip the archive on the Raspberry Pi board and perform the following command:

   ```
   $chmod -R 755 appdb bin
   ```

5. `cd` to the `bin` directory. The contents of the `bin` directory are shown in Figure 1–1.

Figure 1–1 Raspberry Pi Bin Directory



```
Last login: Sat Sep 28 17:18:08 2013

pi@raspberrypi ~ $ cd bin
pi@raspberrypi ~/bin $ ls
installMidlet.sh    listMidlets.sh   runMidlet   runSuite.sh
jwc_properties.ini  removeMidlet.sh  run.sh      usertest.sh
pi@raspberrypi ~/bin $
```

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 Oracle Java ME Embedded on Raspberry Pi can be configured 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
```

2 Installing and Running Applications on the Raspberry Pi Board

Developers can run and debug IMlets on the Raspberry Pi board directly from the NetBeans IDE 8.0 Patch 1 or using the Oracle Java ME SDK 8.0. This chapter describes how to add the board to the Device Connections Manager in the Oracle Java ME SDK 8.0 and how to debug an IMlet on the board from the NetBeans IDE 8.0.

Using the Java Runtime on the Raspberry Pi

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

1. Directly run commands using an LXTerminal command-line shell interface or logging in using the `ssh` protocol.
2. Manually start a Developer Agent program on the desktop host and run commands using the Application Management System (AMS).
3. Run NetBeans IDE 8.0.

Running IMlets on Raspberry Pi Using the Command Shell

If you want to run IMlets directly on the Raspberry Pi board without a Developer Agent program, then you can use the commands shown in Table 2–1.

Table 2–1 Raspberry Pi Shell 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.
`sudo 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 the Oracle Java ME Embedded command-line interface (CLI) and references in this chapter, is synonymous with *MIDlet*.

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 board. Note that the `runSuite.sh` shell command must be preceded by the `sudo` request to ensure that the command can run with superuser privileges and access all the peripherals on the board. Note also that either all commands must be preceded by the `sudo` request or none of them. 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 $ sudo ./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 $ sudo ./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 `sudo 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 $ sudo ./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 $ sudo ./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 $ sudo ./listMidlets.sh
Java is starting. Press Ctrl-C to exit
No suites installed
```

Starting the Developer Agent Program on the Desktop

If you want to use the Developer Agent program, then extract files from the copy of the Oracle Java ME Embedded ZIP archive on the Windows desktop and delete the `/appdb` and the `/bin` directories. The Developer Agent program is a JAR file inside the `util` directory of the Oracle Java ME Embedded distribution, and is named `proxy.jar`. You can start the Developer Agent program on the desktop host computer either in a server or a client mode. After the Developer Agent program starts, use the AMS CLI.

Server Mode Connection

A server mode is used by default. In this mode, start the Java runtime on the Raspberry Pi board to allow access to the AMS, then start the Developer Agent program. You must start the Java runtime on the Raspberry Pi board with the `usertest.sh` command in the command-line shell interface. Run the command using the `sudo` program to obtain superuser privileges to access all the peripherals on the board.

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

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

2. Change to the `util` directory on your desktop host and enter the following command. You should see an output similar to the following:

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

Client Mode Connection

To switch to a client mode connection, perform the following steps.

1. Edit the `jwc_properties.ini` file in the `bin` directory on the Raspberry Pi board as follows:

   - Set the `proxy_connection_mode` property to the `client` value.
   - Set the `proxy.client_connection_address` property to the IP address of the host running the Developer Agent program.

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

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

3. Change to the `util` directory on your desktop host and enter the following command. You should see an output similar to the following:

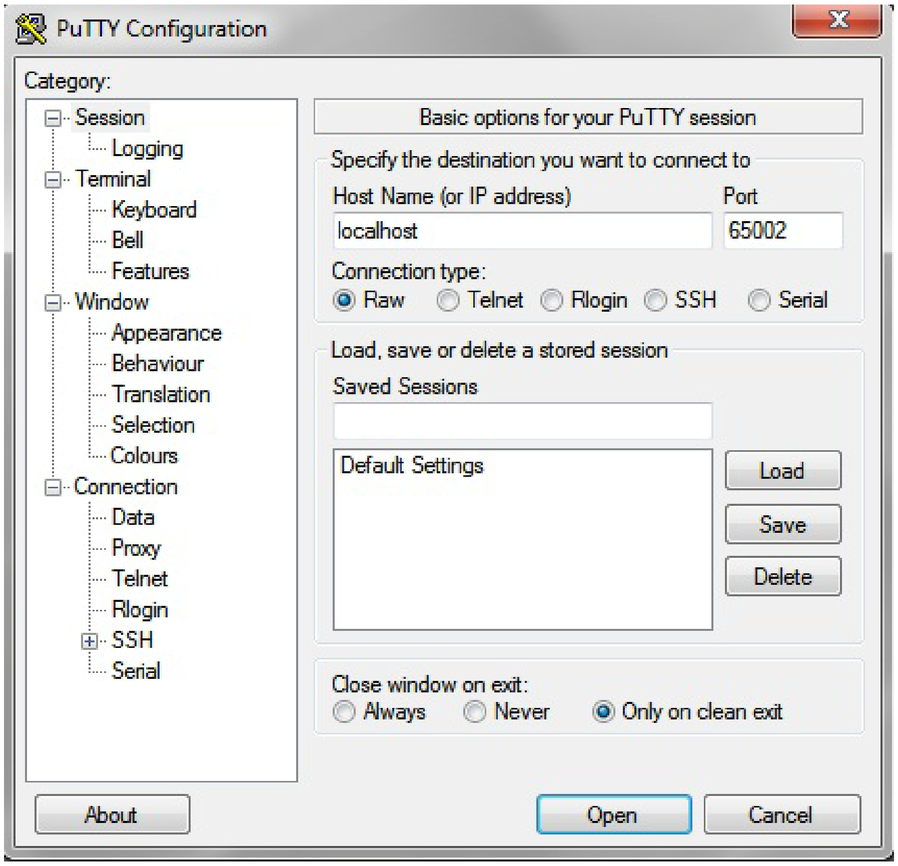
   ```
   C:\mydir\util> java -jar proxy.jar
   Starting with default parameters: -ServerSocketPort 2200 -jdbport 2801
   Channel 8 CLOSED -> AVAILABLE
   Waiting for device connections on port 2200
   ```

Running IMlets on Raspberry Pi Using the AMS CLI

The next step is to make a raw connection to the AMS CLI. Note, however, you must first run the Developer Agent program on the desktop host and the Java runtime - on the Raspberry Pi board as described in Starting the Developer Agent Program on the Desktop unless the Developer Agent was started automatically by Java ME SDK.

At this point, you can start a PuTTY executable file on your desktop computer. Use this to create raw socket connections to the IP address of the host running the Developer Agent, and port 65002. For example, a connection to `localhost` and the port 65002 is shown in Figure 2–1.

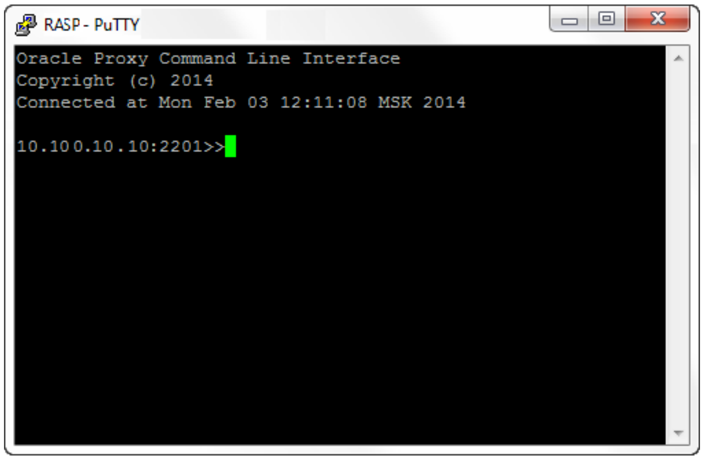
Figure 2–1 Using PuTTY to Connect to the Command-Line Interface



Specify the destination you want to connect to
Host Name (or IP address): localhost
Port: 65002
Connection type: [✓ Raw] Telnet Rlogin SSH Serial
Close window on exit: Always Never [✓ Only on clean exit]

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

Figure 2–2 Command-Line Interface to Raspberry Pi



```
Oracle Proxy Command Line Interface
Copyright (c) 2014
Connected at Mon Feb 03 12:11:08 MSK 2014

10.100.10.10:2201>>
```

Caution: 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.

You can use the command-line interface to run the AMS commands shown in Table 2–2.

Table 2–2 AMS CLI Commands

Syntax	Description
`ams-list [INDEX or NAME|VENDOR]`	List all installed IMlet suites and their status or show the detail of a single suite.
`ams-install <URL> [username:password][hostdownload]`	Install an IMlet using the specified JAR or JAD file, specified as a URL. An optional user name and password can be supplied for login information either in the URL or by the `auth` parameter. When run without the `hostdownload` option, only `http://` URLs must be specified. The `hostdownload` option enables you to install an IMlet using the JAR file specified by the `file://` URL. Note that the JAR file must be located on the host.
`ams-update <INDEX or NAME|VENDOR>`	Update the installed IMlet.
`ams-remove <INDEX or NAME|VENDOR>`	Remove an installed IMlet.
`ams-run <INDEX or NAME|VENDOR> [IMLET_ID] [-debug]`	Run the specified IMlet or the default if none is specified. You can specify optional debug parameter to run the IMlet in debug mode.
`ams-stop <INDEX or NAME|VENDOR> [IMLET_ID]`	Stop the specified IMlet or the default if none is specified.
`ams-info <INDEX or NAME|VENDOR>`	Show information about the installed IMlet.

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
```

You can use the properties commands summarized in Table 2–3 and file system commands summarized in Table 2–4.

Table 2–3 Security and Properties Commands

Syntax	Description
`help [command name]`	List the available commands or detailed usages for a single command.
`properties-list`	Show the list of names of properties which control the Java ME runtime (properties that are set in the `properties.ini` file).
`get-property <NAME> [-i]`	Return a value of the property identified by `<NAME>`.
`set-property <NAME> <VALUE>`	Set a property identified by `<NAME>` with the value `<VALUE>`.
`save-properties`	Save properties to an internal storage.
`blacklist -client <NAME>`	Blacklist clients and applications.
`blacklist -app <NAME|VENDOR>`	

Table 2–4 File System Commands

Syntax	Description
`cd <deviceDirectoryName>`	Change the working directory on the device.
`delete <deviceFileName>`	Delete a file on the device.
`get <deviceFileName> <hostFileName>`	Copy a file from the device to the host.

Table 2–4 (Cont.) File System Commands

Syntax	Description
`ls [<deviceDirectoryName>]`	Display a list of files and subdirectories in a device directory. In a result listing, subdirectories are marked with a trailing file separator symbol, such as "\" on Windows and "/" on the Raspberry Pi board.
`mkdir <deviceDirectoryName>`	Create a directory on the device.
`pwd`	Write the current working directory on the device.
`put <hostFileName> <deviceFileName>`	Copy a local host file to the device.

The CLI supports working with multiple devices. You can use the device commands summarized in Table 2–5.

Table 2–5 Device Commands

Syntax	Description
`device-list`	List all connected devices.
`device-change <INDEX>`	Make the specified device current.
`shutdown [-r]`	Perform either a shutdown of the board or a reboot if the `-r` parameter has been passed.
`exit`	Terminate the current CLI session.

You can use the keystore commands summarized in Table 2–6.

Table 2–6 Keystore Commands

Syntax	Description
`ks-delete (-owner <owner name> | -number <key number>)`	Delete a key from a ME store.
`ks-export -number <key number> -out <full file name>`	Export a key from a device keystore by index.
`ks-import [-keystore <filename>] [-storepass <password>] [-keypass <password>] [-alias <key alias>]`	Import a public key from a JCE keystore into a ME keystore.
`ks-list`	List the owner and validity period of each key in a ME keystore.

Using NetBeans with the Raspberry Pi Board

Running and debugging IMlet projects on the Raspberry Pi board using the NetBeans IDE 8.0 requires the following software:

- NetBeans IDE 8.0 with Java ME 8 support
- Oracle Java ME SDK 8.0
- Oracle Java ME SDK 8.0 plugins

For complete instructions about installing Oracle Java ME SDK 8.0, the NetBeans IDE 8.0, and Oracle Java ME SDK 8.0 plug-ins for NetBeans, see *Oracle Java ME SDK Developer's Guide*.

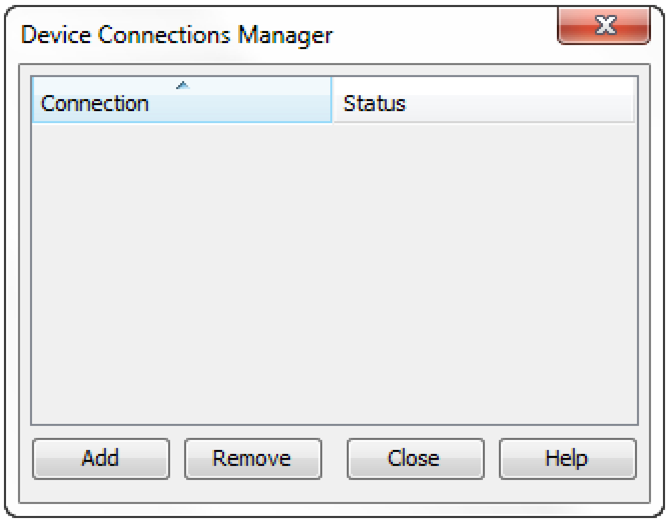
Note: This chapter assumes that the Raspberry Pi board is already set up and connected to the Windows platform running Oracle Java ME SDK 8.0 and that NetBeans IDE 8.0 has already been started.

Adding the Raspberry Pi Board to the Device Connection Manager

Follow these steps to add the Raspberry Pi board to the Device Connections Manager in Oracle Java ME SDK 8.0:

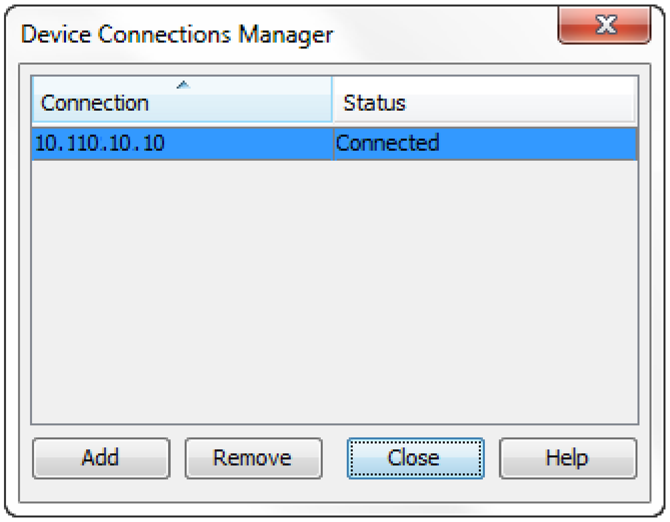
1. Ensure that the `sudo. /usertest.sh` script in the `/bin` directory is running on the Raspberry Pi board.
2. Ensure that the Developer Agent program does not run on the desktop computer.
3. Start the Oracle Java ME SDK 8.0 Device Connections Manager (located at <*SDK Installation Folder*>`/bin/device-manager.exe`) and click its icon in the taskbar. A Device Connections Manager window is shown in Figure 2–3.

Figure 2–3 Device Connections Manager Window



4. Click the **Add** 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



Connection	Status
10.110.10.10	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 with an IMlet you want to run or debug.
- Creating a new NetBeans project.

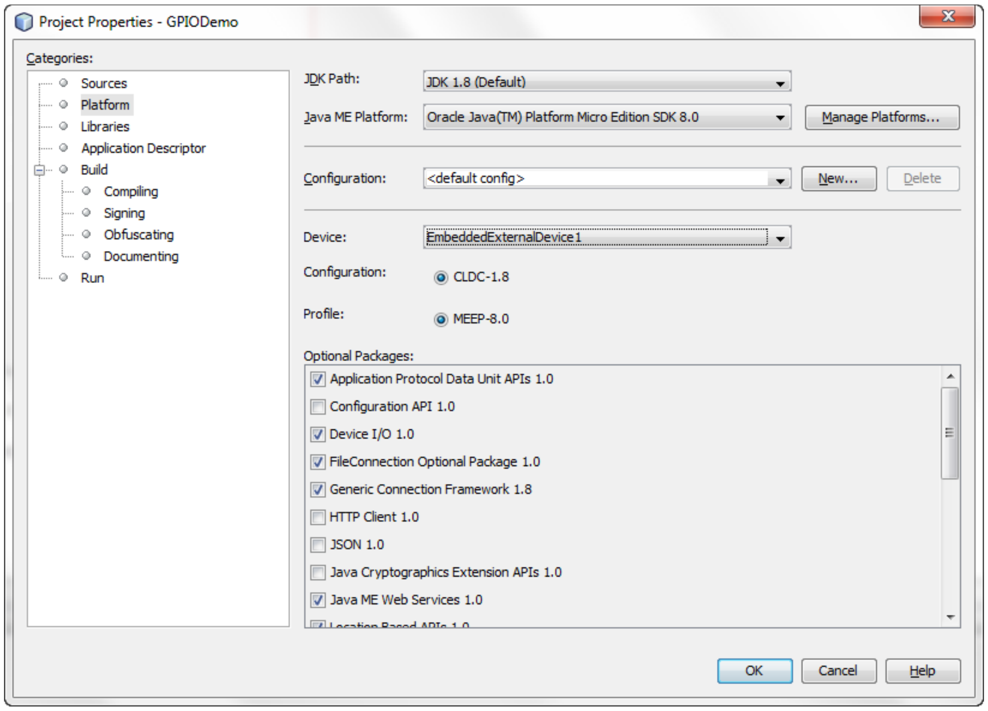
After you assign the board to your project, clicking **Run Project** in the NetBeans IDE 8.0 runs your IMlets on the board instead of on the emulator.

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. Select the Platform Type (CLDC) and ensure that **Oracle Java(TM) Platform Micro Edition SDK 8.0** is selected in the Java ME Platform list.
4. Select **EmbeddedExternalDevice1** 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



Project Properties - GPIODemo

Categories:
- Sources
- Platform
- Libraries
- Application Descriptor
- Build
  - Compiling
  - Signing
  - Obfuscating
  - Documenting
- Run

JDK Path: JDK 1.8 (Default)

Java ME Platform: Oracle Java(TM) Platform Micro Edition SDK 8.0

Configuration: <default config>

Device: EmbeddedExternalDevice1

Configuration: ● CLDC-1.8

Profile: ● MEEP-8.0

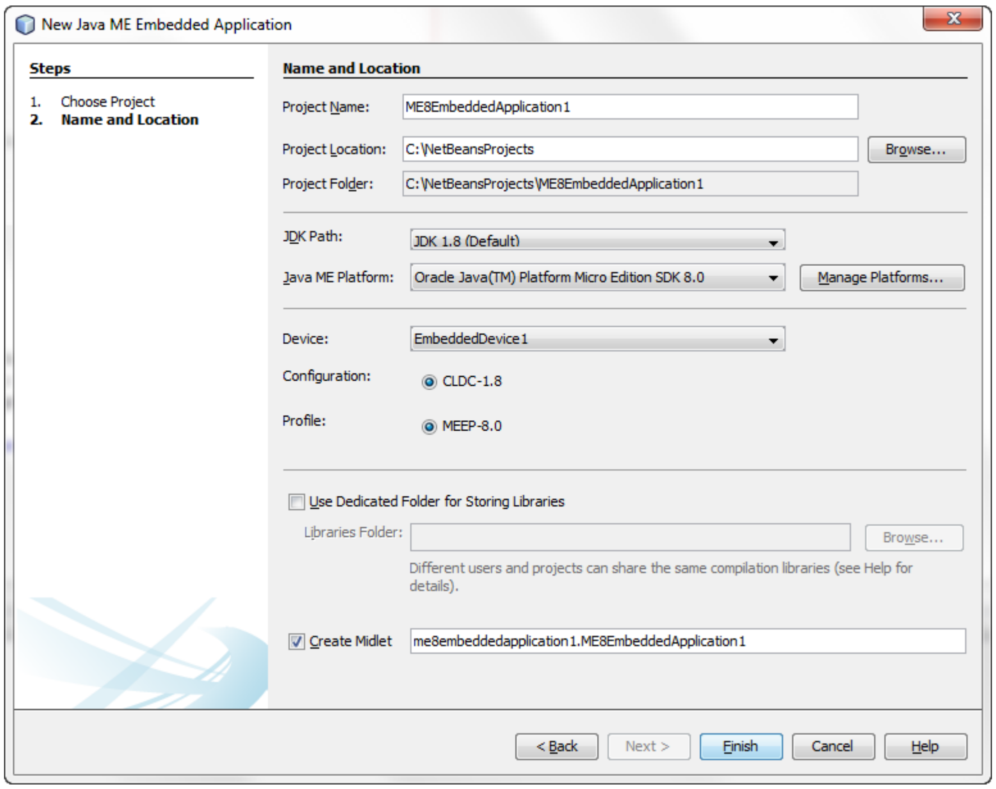
Optional Packages:
- [✓] Application Protocol Data Unit APIs 1.0
- [ ] Configuration API 1.0
- [✓] Device I/O 1.0
- [✓] FileConnection Optional Package 1.0
- [✓] Generic Connection Framework 1.8
- [ ] HTTP Client 1.0
- [ ] JSON 1.0
- [ ] Java Cryptographics Extension APIs 1.0
- [✓] Java ME Web Services 1.0
- [✓] Location Based APIs 1.0

Creating a New NetBeans Project

If you are creating a new NetBeans project, follow these steps:

1. Select **File** then **New Project**.
2. Select the **Java ME Embedded** category and **Java ME 8 Embedded Application** in Projects pane. Click **Next**.
3. Provide a project name, for example, `ME8EmbeddedApplication1`. Ensure that the Java ME Platform is **Oracle Java(TM) Platform Micro Edition SDK 8.0** and the **Create Midlet** option is selected.
4. Select **EmbeddedExternalDevice1** from the Device drop-down list and click **Finish**, as shown in Figure 2–6.

Figure 2–6 Creating a New Project


New Java ME Embedded Application

Steps
1. Choose Project
2. **Name and Location**

Name and Location

Project Name: ME8EmbeddedApplication1

Project Location: C:\NetBeansProjects

Project Folder: C:\NetBeansProjects\ME8EmbeddedApplication1

JDK Path: JDK 1.8 (Default)

Java ME Platform: Oracle Java(TM) Platform Micro Edition SDK 8.0

Device: EmbeddedDevice1

Configuration: CLDC-1.8

Profile: MEEP-8.0

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

☑ Create Midlet me8embeddedapplication1.ME8EmbeddedApplication1

When the new project is created, it is displayed in NetBeans IDE with the name `ME8EmbeddedApplication1`.

Sample Source Code

Now you can update the generic project that you created with the sample code shown in the following example. This sample application obtains an object representing GPIO pin 2 from the `DeviceManager` object, and tries to obtain its high/low value.

```
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) {
    }
}
```

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 sample code and paste it in the Source window.
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 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.

Accessing 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 1.0* specification at:

http://docs.oracle.com/javame/8.0/api/dio/api/index.html

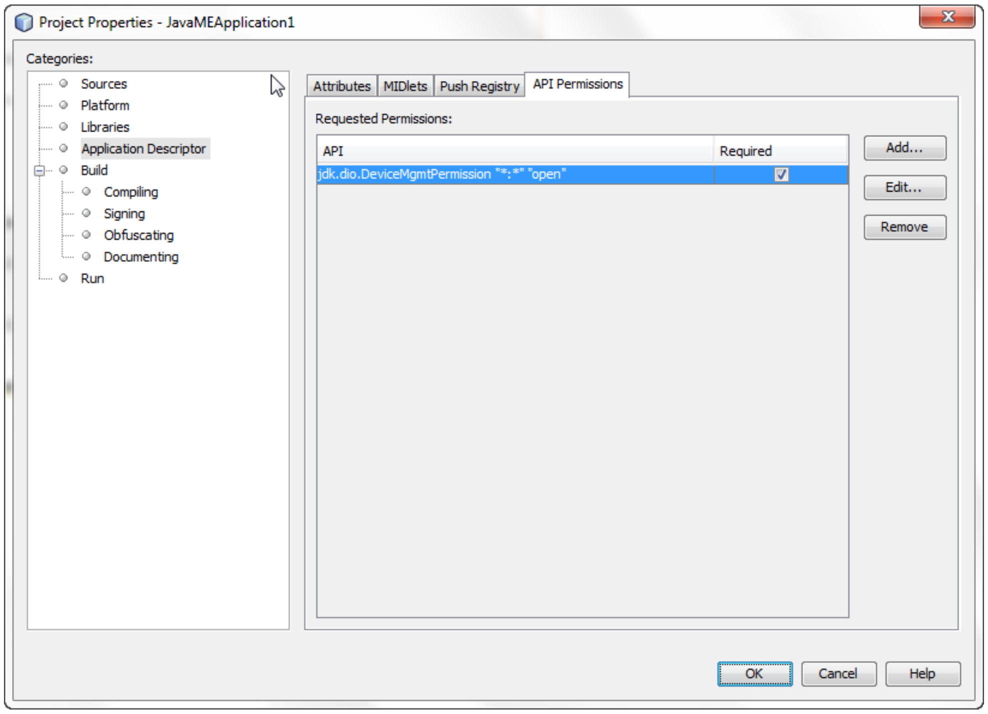
Signing the Application with API Permissions

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 2–7.
4. Click **OK** to close the project properties dialog.

Figure 2–7 Adding API Permissions with NetBeans



API	Required
jdk.dio.DeviceMgmtPermission "*:*" "open"	✓

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

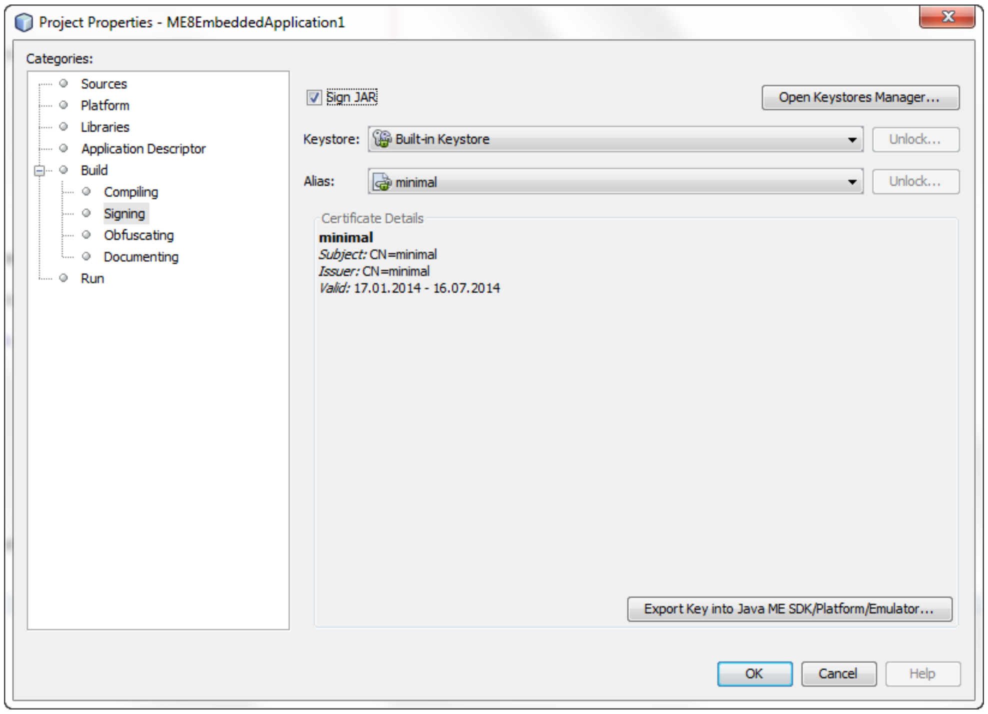
```
MIDlet-Permission-1: jdk.dio.DeviceMgmtPermission "*:*" "open"
```

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 2–8.

Figure 2–8 Signing Application JAR with NetBeans



☑ Sign JAR

Keystore: Built-in Keystore

Alias: minimal

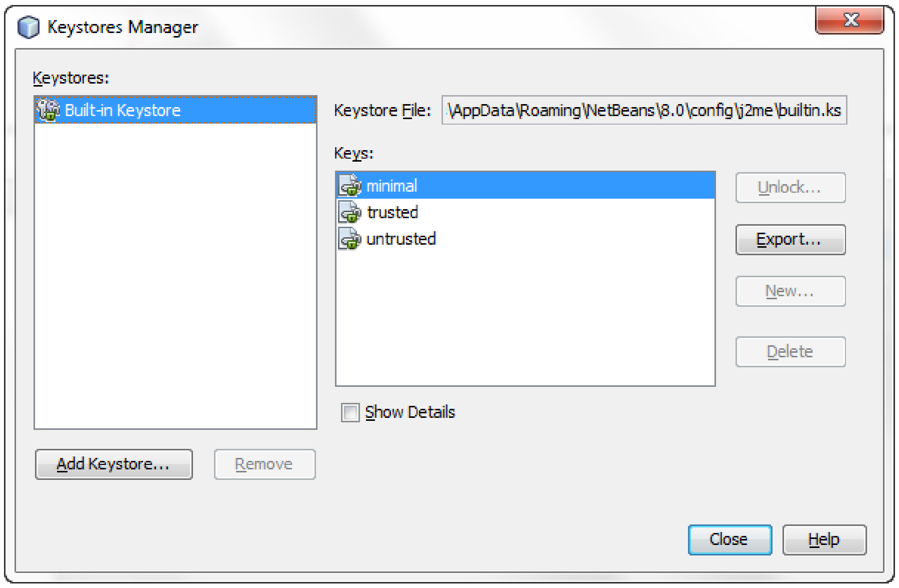
minimal
Subject: CN=minimal
Issuer: CN=minimal
Valid: 17.01.2014 - 16.07.2014

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 2–9.

Figure 2–9 Keystores Manager Window



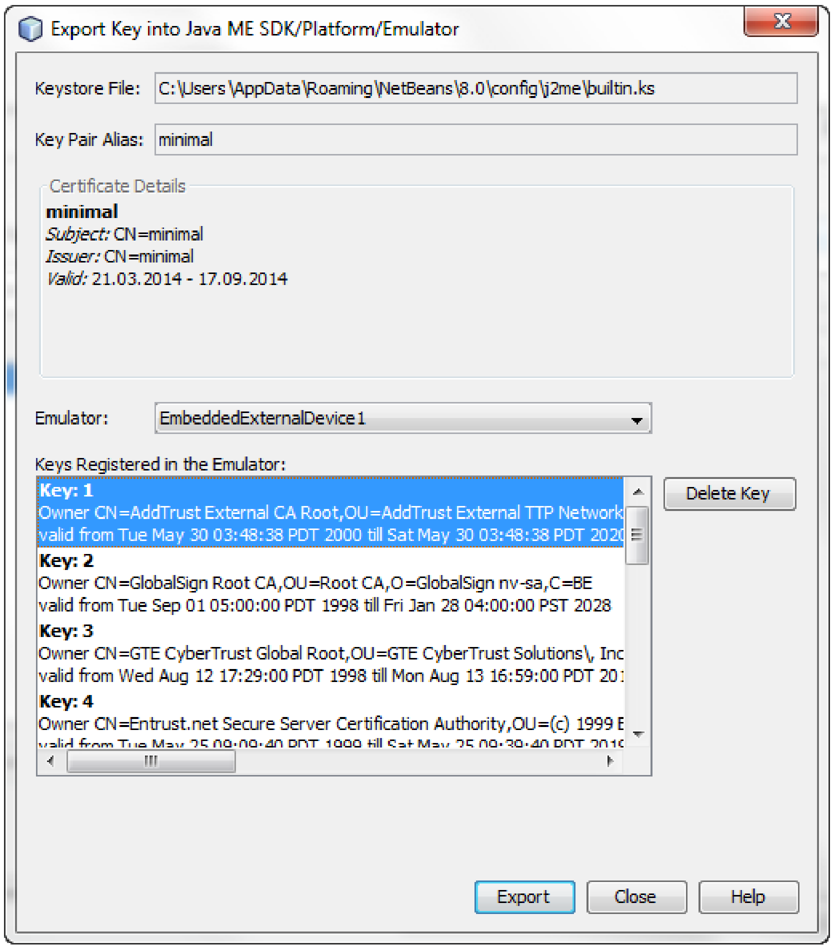
Keystores: Built-in Keystore

Keystore File: \AppData\Roaming\NetBeans\8.0\config\j2me\builtin.ks

Keys:
minimal
trusted
untrusted

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 2–10.

Figure 2–10 Exporting Key on a Device


Export Key into Java ME SDK/Platform/Emulator

Keystore File: C:\Users\AppData\Roaming\NetBeans\8.0\config\j2me\builtin.ks

Key Pair Alias: minimal

Certificate Details

minimal
Subject: CN=minimal
Issuer: CN=minimal
Valid: 21.03.2014 - 17.09.2014

Emulator: EmbeddedExternalDevice1

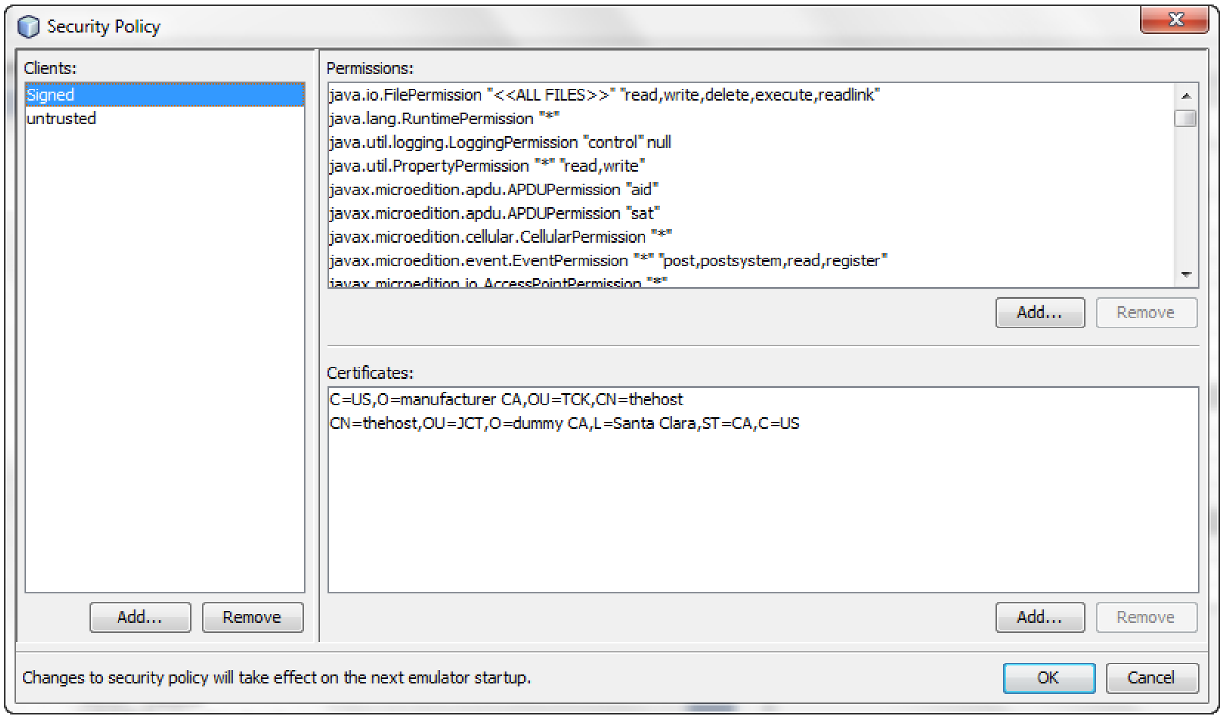
Keys Registered in the Emulator:

```
Key: 1
Owner CN=AddTrust External CA Root,OU=AddTrust External TTP Network
valid from Tue May 30 03:48:38 PDT 2000 till Sat May 30 03:48:38 PDT 2020
Key: 2
Owner CN=GlobalSign Root CA,OU=Root CA,O=GlobalSign nv-sa,C=BE
valid from Tue Sep 01 05:00:00 PDT 1998 till Fri Jan 28 04:00:00 PST 2028
Key: 3
Owner CN=GTE CyberTrust Global Root,OU=GTE CyberTrust Solutions\, Inc
valid from Wed Aug 12 17:29:00 PDT 1998 till Mon Aug 13 16:59:00 PDT 201
Key: 4
Owner CN=Entrust.net Secure Server Certification Authority,OU=(c) 1999 E
valid from Tue May 25 09:09:40 PDT 1999 till Sat May 25 09:39:40 PDT 2019
```

7. Select **Tools** then **Java ME** and then **Device Selector**. The Device Selector window opens.

8. In the Device Selector window, right-click **EmbeddedExternalDevice1** and select **Edit Security Policy**. This opens the Security Policy window shown in Figure 2–11.

Figure 2–11 Editing Security Policy


Security Policy

Clients:

Signed
untrusted

Permissions:

```
java.io.FilePermission "<<ALL FILES>>" "read,write,delete,execute,readlink"
java.lang.RuntimePermission "*"
java.util.logging.LoggingPermission "control" null
java.util.PropertyPermission "*" "read,write"
javax.microedition.apdu.APDUPermission "aid"
javax.microedition.apdu.APDUPermission "sat"
javax.microedition.cellular.CellularPermission "*"
javax.microedition.event.EventPermission "*" "post,postsystem,read,register"
javax.microedition.io.AccessPointPermission "*"
```

Certificates:

```
C=US,O=manufacturer CA,OU=TCK,CN=thehost
CN=thehost,OU=JCT,O=dummy CA,L=Santa Clara,ST=CA,C=US
```

Changes to security policy will take effect on the next emulator startup.

9. Ensure that the certificate is associated with the security client. Select the security client or click **Add** to add the security client.

10. For the selected security client, ensure that the certificate with the specified common name (CN) is listed on the list of certificates. If not, click **Add** to add the certificate.

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 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.

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
```

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

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.

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 will 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.

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. See the following site for more information:

  https://developer.qualcomm.com/mobile-development/development-devices-boards/development-boards/internet-of-everything-development-platform/tools-and-resources
- **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` does not support device names. Do not use the `DeviceConfig.HardwareAddressing.getDeviceName()` method.
- The present implementation utilizes the I2C bus STOP-START sequence instead of REPEATED START, keeping a combined message uninterrupted by another transaction on the same I2C bus.
- 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.

DeviceI ID	Device Name	Mapped	Configuration

2	GPIO2	GPIO 2	`controllerNumber = 0`
			`pinNumber = 2`
			direction = GPIOPinConfig.DIR_INPUT_ ONLY
			`mode = GPIOPinConfig.MODE_INPUT_ PULL_UP`
			`trigger = GPIOPinConfig.TRIGGER_ BOTH_EDGES`
			`initValue` - ignored

3	GPIO3	GPIO 3	`controllerNumber = 0`
			`pinNumber = 3`
			`direction = GPIOPinConfig.DIR_INPUT_ ONLY`
			`mode = GPIOPinConfig.MODE_INPUT_ PULL_UP`
			`trigger = GPIOPinConfig.TRIGGER_ BOTH_EDGES`
			`initValue` - ignored

4	GPIO4	GPIO 4	`controllerNumber = 0`
			`pinNumber = 4`
			`direction = GPIOPinConfig.DIR_INPUT_ ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_ BOTH_EDGES`
			`initValue` - ignored

7	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`

Devicel ID	Device Name	Mapped	Configuration

8	GPIO8	GPIO 8	`controllerNumber = 0`
			`pinNumber = 8`
			`direction = GPIOPinConfig.DIR_OUTPUT_ONLY`
			`mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL`
			`trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES`
			`initValue = false`

9	GPIO9	GPIO 9	`controllerNumber = 0`
			`pinNumber = 9`
			`direction = GPIOPinConfig.DIR_INPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES`
			`initValue` - ignored

10	GPIO10	GPIO 10	`controllerNumber = 0`
			`pinNumber = 10`
			`direction = GPIOPinConfig.DIR_INPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES`
			`initValue` - ignored

11	GPIO11	GPIO 11	`controllerNumber = 0`
			`pinNumber = 11`
			`direction = GPIOPinConfig.DIR_INPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES`
			`initValue` - ignored

14	GPIO14	GPIO 14	`controllerNumber = 0`
			`pinNumber = 14`
			`direction = GPIOPinConfig.DIR_OUTPUT_ONLY`
			`mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL`
			`trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES`
			`initValue = false`

DeviceI ID	Device Name	Mapped	Configuration

15	GPIO15	GPIO 15	`controllerNumber = 0`
			`pinNumber = 15`
			`direction = GPIOPinConfig.DIR_OUTPUT_ONLY`
			`mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL`
			`trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES`
			`initValue = false`

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

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

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

23	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`

DeviceI ID	Device Name	Mapped	Configuration

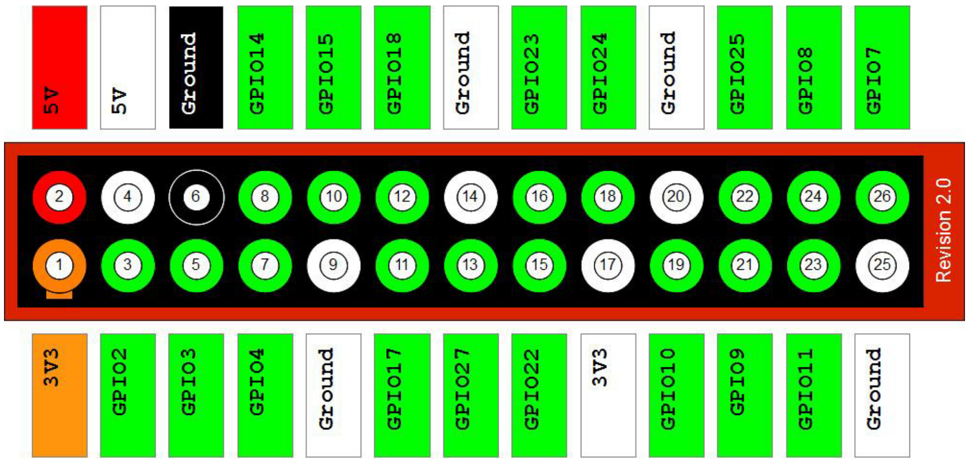
24	GPIO24	GPIO 24	`controllerNumber = 0`
			`pinNumber = 24`
			`direction = GPIOPinConfig.DIR_OUTPUT_ONLY`
			`mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL`
			`trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES`
			`initValue = false`

25	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`

27	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.
- To work with GPIO, you must run Java as the root superuser.
- 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 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 `clockFrequency` field is ignored.
- 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
```

- I2C can be used without administrative rights. To do so, you should have owner or group rights to files `/dev/i2c-*`. This can easily done by installing the `i2c-tools` package (`$ sudo apt-get install i2c-tools`) and adding the `pi` user to the `i2c` group (`$ sudo adduser pi i2c`). Alternatively, you can use udev's rules.

MMIO

The following MMIO peripherals are available:

Device ID	Device Name	Mapped	Configuration
61	PWM		

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

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
12	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: `$sudo 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

40	UART	GPIO 14 (TXD)
		GPIO 15 (RXD)
			`uartNumber = 0`
			`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.

- Only the internal UART controller is supported (`/dev/ttyAMA0` for revision 2). Consequently, 0 is the only permissible value for the `UARTConfig.uartNumber` parameter.
- By default, the Raspberry Pi uses the UART as a serial console. Before using UART, make sure that `/dev/ttyAMA0` isn't 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`.
- 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.
- The `deviceaccess.uart.prefix` property in the `jwc_properties.ini` file may contain a prefix for easy conversion of the `UARTConfig.portNumber` value to a platform-specific port name. For example, the property may be set to "`COM`" in a Windows environment, or "`/dev/ttyS`" in a Linux environment such that appending on a port number will correctly map to the port name.
- The following parameters are supported in an ad-hoc configuration:
  – `baudRate` - 110, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200
  – `dataBits` - 7, 8
  – `parity` - PARITY_ODD, PARITY_EVEN, PARITY_NONE
  – `stopBits` - 1, 2
  – `flowcontrol` - FLOWCONTROL_NONE

Watchdog

The following watchdog devices are pre-configured:

Device ID	Device Name	Mapped	Configuration
30	WDG	Platform Watchdog	
31	WWDG		

ATCmdDevice

The following ATCmd device is preconfigured:

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

B

Configuring the Java Runtime Properties

There are several ways to change the value of a property that affects Java's configuration or behavior at runtime.

Direct Modification of 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.