Oracle® Java ME Embedded

Getting Started Guide for the Reference Platform (Raspberry Pi)

Release 3.3

E38384-01

June 2013

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

ORACLE

Oracle Java ME Embedded Getting Started Guide for the Reference Platform (Raspberry Pi), Release 3.3

E38384-01

Contents

3 Troubleshooting

A Raspberry Pi Board Peripheral List

B AMS Installer Error Codes

List of Examples

List of Figures

List of Tables

Preface

This book describes how to install Oracle Java ME Embedded software onto a Raspberry Pi embedded device. Readers using this guide should be familiar with the *Information Module Profile - Next Generation (IMP-NG) 1.0 Specification*.

Audience

This document is intended for developers who want to run Oracle Java ME Embedded software on embedded devices.

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 with the Oracle Java ME Embedded software, see the Release Notes.

Operating System Commands

This document does not contain information on basic commands and procedures such as opening a terminal window, changing directories, and setting environment variables. See the software documentation that you received with your system for this information.

Shell Prompts

Shell	Prompt
Bourne shell and Korn shell	`$`
Windows	*directory*>

Conventions

The following text conventions are used in this document:

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

1

Running on the Raspberry Pi Board

This chapter describes installing the Oracle Java ME Embedded software on the Raspberry Pi board, configuring the 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:

- The Raspberry Pi Rev. B 512 MB Board
- A micro-USB power supply of .7 or greater amps, and 5 volts. Note that the power supply must have a *micro*-USB connector, not a regular USB or mini-USB connector.
- A USB keyboard and mouse, as well as a monitor. If necessary for your monitor, an HDMI to DVI video cable or adapter.
- An SD card of 2 GB or greater. An SD-HC class 10 card is recommended. Do not use a high speed SD card, as it may be too fast for the Raspberry Pi board.
- An ethernet cable with an RJ-45 connection, and a connection to a network with a DHCP server.
- A terminal emulation 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 is directly downloadable as `putty.exe`. The terminal emulator is used to connect to the AMS command-line interface (CLI) that issues commands to the board.

Preparing the Raspberry Pi Board

To develop on the Raspberry Pi board, you must first download and install the Wheezy variant of Raspbian Debian Linux on the Pi board. If you have not already done this, use the following steps:

1. Download the Raspbian "Wheezy" hard-float (Debian Linux) 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 akin to "burning" a disk image onto a CD-ROM or DVD-ROM. There are a number of utilities that will perform this action. With Windows, for example, you can use the Disk Image Writer utility located at `https://launchpad.net/win32-image-writer` to perform this task. For Mac, 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.
7. Connect power to the Pi. The red light on the Raspberry Pi should glow, then in a few seconds, the green light should blink. The blinking green light indicates that the Raspberry Pi is booting Linux.
8. If the Linux installation was successful, the Raspberry Pi will boot and obtain a DHCP address.
9. Next, a configuration program (`raspi-config`) runs, which helps you expand the filesystem partition on the SD card, configure the keyboard, timezone, reset the default password, and several other useful system items. Use the up-down arrow keys to make a menu choice. Use the left-right arrow keys to select `OK` or `Cancel`. Press `Return` to execute your choice. Note that the default user name is "`pi`", and the default password is "`raspberry`".
10. If you wish, perform an update, start the ssh server, and set the graphical desktop to automatically start, then press `Finish`. At this point, the board should reboot. Login if necessary, and if you're using the desktop, start a LXTerminal. Then, run the `ifconfig` command to display the Pi's 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 Java ME Embedded Software

Use an `sftp` client or `scp` command to transfer the Oracle Java ME Embedded ZIP file to the Raspberry Pi. 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; it is available from the same address as the PuTTY executable:

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

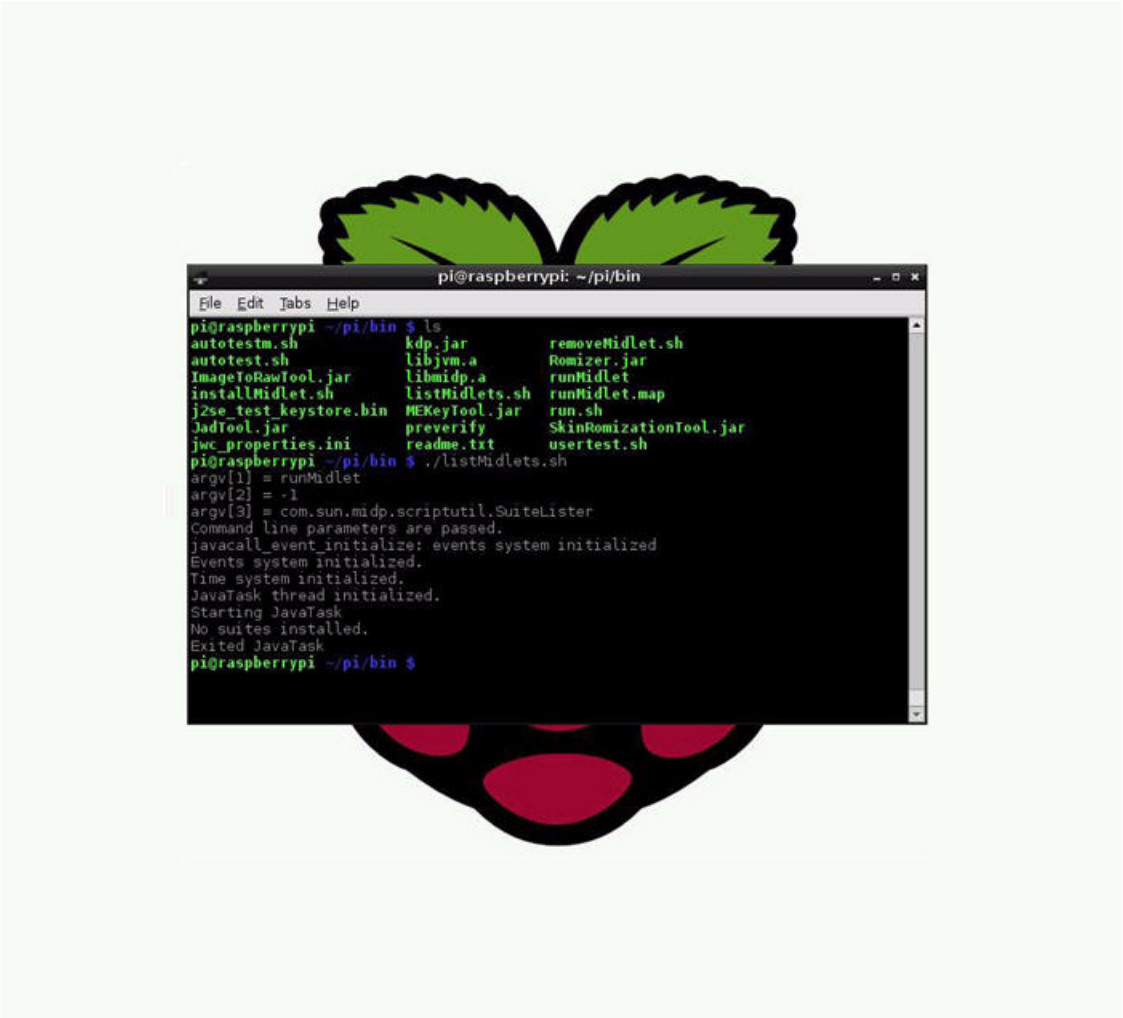
Once the file is transferred, either go directly to the keyboard and the mouse connected to the Raspberry Pi, or start a secure shell script on your desktop to connect to the board using the following command:

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

Change directory to the location that you uploaded the Oracle Java ME Embedded distribution, create a new folder, and unzip the distribution to a folder.

At this point, you see three subdirectories: `appdb`, `bin`, and `lib`. The contents of the `bin` directory are the most important, and from the desktop on the Raspberry Pi, are shown in Figure 1–1.

Figure 1–1 Raspberry Pi Directory Display



There are two ways to interact with the Oracle Java ME Embedded. The first is to directly execute commands using the LXTerminal command-line shell interface or logging in via `ssh`. The second is to connect to the board using port 65002 and run commands using the Application Management System (AMS).

Adding a Proxy for Network Connections

If a proxy server is required for network connections (such as HTTP or apt-get), the server can be configured for IMlets 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
```

Running IMlets on the Raspberry Pi Using the Command Shell

You can use a command-line shell interface to run the commands shown in Table 1–1.

Table 1–1 Raspberry Pi Shell Commands

Syntax	Description
`listMidlets.sh [SUITE_ID or NAME]`	List all installed IMlet suites and their statuses 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]`	Execute the specified IMlet or the default if none is specified. All logging information from the IMlet appears in the stdout of this command.

The following is a typical example of using the commands to install, list, run, and remove a Java ME Embedded application on the Raspberry Pi board. Note that the `runSuite` shell command is preceded by the `sudo` request to ensure that the command can run with superuser privileges and access all the peripherals on the 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 filesystem.

```
pi@raspberrypi ~/pi/bin $ ./installMidlet.sh EmbeddedTestProject.jar
argv[1] = runMidlet
argv[2] = -1
argv[3] = com.sun.midp.scriptutil.CommandLineInstaller
argv[4] = I
argv[5] = EmbeddedTestProject.jar
Command line parameters are passed.
javacall_event_initialize: events system initialized
Events system initialized.
Time system initialized.
JavaTask thread initialized.
Starting JavaTask
The suite was successfully installed, ID: 2
Exited JavaTask
```

If the install process shows any error code, see Table B–1 in Appendix B, "AMS Installer Error Codes" for more information on how to resolve the error.

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

```
pi@raspberrypi ~/pi/bin $ ./listMidlets.sh
argv[1] = runMidlet
argv[2] = -1
argv[3] = com.sun.midp.scriptutil.SuiteLister
Command line parameters are passed.
javacall_event_initialize: events system initialized
Events system initialized.
Time system initialized.
JavaTask thread initialized.
Java is starting. Press Ctrl-C to exit
Starting JavaTask
Suite: 2
  Name: EmbeddedTestProject
```

```
  Version: 1.0
  Vendor: Vendor
  MIDlets:
    IMlet: GPIODemo
Exited JavaTask
```

The `sudo runMidlet` command can be used to execute any installed IMlet. This command will execute the IMlet that was just installed, passing any logging information to the stdout of this command. Note that you can press the Ctrl-C key to exit from this command, which will terminate the app.

```
pi@raspberrypi ~/pi/bin $ sudo ./runSuite.sh GPIODemo
argv[1] = GPIODemo
Command line parameters are passed.
javacall_event_initialize: events system initialized
Events system initialized.
Time system initialized.
JavaTask thread initialized.
Java is starting. Press Ctrl-C to exit
Starting JavaTask
Watchdog is starting
Network: Init
App started
```

The `removeMidlet.sh` command can be used to remove any installed IMlet.

```
pi@raspberrypi ~/pi/bin $ ./removeMidlet.sh 2
argv[1] = runMidlet
argv[2] = -1
argv[3] = com.sun.midp.scriptutil.SuiteRemover
argv[4] = 2
Command line parameters are passed.
javacall_event_initialize: events system initialized
Events system initialized.
Time system initialized.
JavaTask thread initialized.
Java is starting. Press Ctrl-C to exit
Starting JavaTask
Suite removed
Exited JavaTask
pi@raspberrypi ~/pi/bin $
```

The results can again be verified with the `listMidlets.sh` command.

```
pi@raspberrypi ~/pi/bin $ ./listMidlets.sh
argv[1] = runMidlet
argv[2] = -1
argv[3] = com.sun.midp.scriptutil.SuiteLister
Command line parameters are passed.
javacall_event_initialize: events system initialized
Events system initialized.
Time system initialized.
JavaTask thread initialized.

Java is starting. Press Ctrl-C to exit
Starting JavaTask
No suites installed
Network: Init
Exited JavaTask
```

Running IMlets on the Raspberry Pi Using the AMS CLI

The second method of connecting to the Raspberry Pi board is to make a raw connection to the AMS CLI on port 65002 of the board. However, in order for this to work, you must first start the Application Management System (AMS) on the board with the `usertest.sh` command in the command-line shell interface. Note that the command must be run using `sudo` to obtain superuser privileges to access all the peripherals on the board.

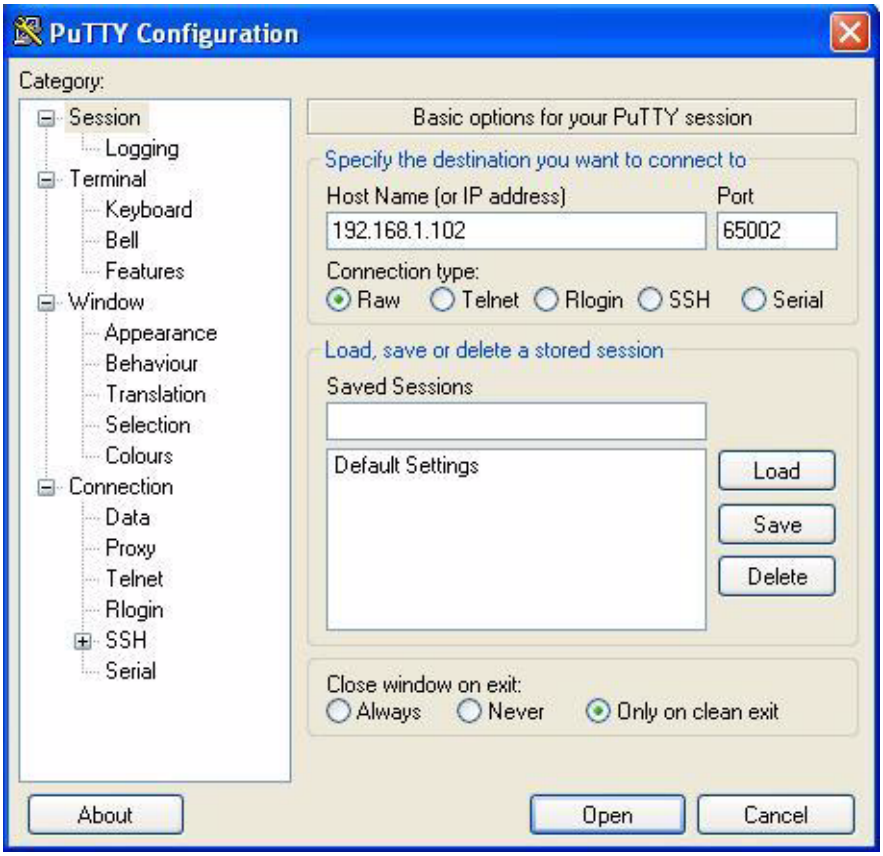
Note: All logging information from IMlets appears in the stdout of this command.

---

pi@raspberrypi ~/pi/bin $ **sudo ./usertest.sh**

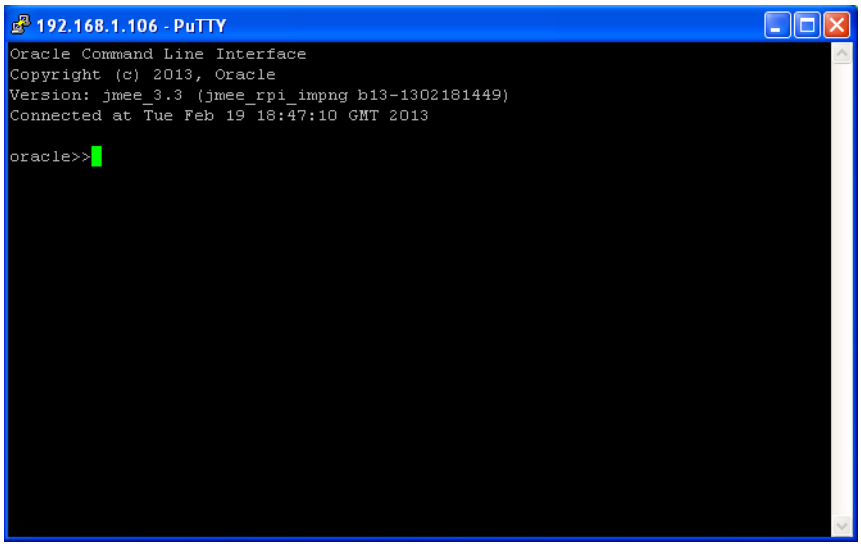
Next, start a PuTTY executable on your desktop computer. Use this to create raw socket connections to the IP address of the Raspberry Pi board, and port 65002. For example, a connection to the IP address of 192.168.1.102 and the port 65002 is shown in Figure 1–2. Note that the IP address of your Raspberry Pi board may be different.

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



The window from port 65002 provides a command-line interface (CLI), and is shown in Figure 1–3:

Figure 1–3 Command-Line Interface to the Raspberry Pi



WARNING: The command-line interface (CLI) feature in this Oracle Java ME Embedded software release is provided only as a concept for your reference. It uses insecure connections with no encryption, authentication, or authorization.

You can use the command-line interface to run the AMS commands shown in Table 1–2.

Table 1–2 AMS CLI Commands

Syntax	Description
`ams-list [INDEX or NAME|VENDOR]`	List all installed IMlet suites and their statuses or show the detail of a single suite
`ams-install <URL> [username:password]`	Install an IMlet using the specified JAR or JAD file, specified as a URL. An optional username and password can be supplied for login information as well.
`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]`	Execute the specified IMlet or the default if none is specified. An optional debug parameter can be specified 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-suspend <INDEX or NAME|VENDOR> [IMLET_ID]`	Suspend (pause) the specified IMlet or the default if none is specified
`ams-resume <INDEX or NAME|VENDOR> [IMLET_ID]`	Resume the specified IMlet or the default if none is specified

Table 1–2 (Cont.) AMS CLI Commands

Syntax	Description
`ams-setup <INDEX or NAME|VENDOR>`	Display the setup menu of the IMlet
`ams-info <INDEX or NAME|VENDOR>`	Show information about the installed IMlet
`ams-log <command> [param1, param2, ..., paramN]` `ams-log wdog`	Display the IMlet log or watchdog log if recorded by the watchdog handler in the platform
`ams-logger-list [INDEX or NAME|VENDOR]`	Retrieve the logger list for the IMlet or all the tasks if one is not specified
`ams-logger-info <INDEX or NAME|VENDOR> [LOGGER_NAME]`	Retrieve logger info for the specified IMlet and logger or all the loggers if is one is not specified
`ams-logger-level-set <INDEX or NAME|VENDOR> [LOGGER_NAME] <LOGGER_LEVEL>`	Set the logger level for specified IMlet or all loggers if one is not specified
`help [command name]`	List the available commands or detailed usages for a single command
`sysmenu <on PASSWORD|off>`	Enable hidden system menu commands. Currently, the password is `12345`.
`exit`	Terminates the current session.

When the `sysmenu` command is entered with the on option, additional system menu commands are available with the AMS CLI, as shown in Table 1–3.

Table 1–3 Additional System Commands Available in the AMS CLI

Syntax	Description
`setprop <KEY> <VALUE>`	Sets a property identified by `<KEY>` with the value `<VALUE>`
`getprop <KEY>`	Returns a property identified by `<KEY>`
`odd [on|off]`	Explicitly sets the on-device debugging (ODD) property to on or off. If no parameters are passed, returns the current ODD value.
`shutdown [-r]`	Perform either a shutdown of the board, or a reboot if the `-r` parameter has been passed.

Here is a typical example of using the Application Management System (AMS) to install, list, run, and remove a Java ME Embedded application on the board.

```
oracle>> ams-install file:///some/directory/hello.jar
<<ams-install,start install,file:///some/directory/hello.jar
<<ams-install,install status: stage 0, 5%
<<ams-install,install status: stage 3, 100%
<<ams-install,install status: stage 4, 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 0, 5%
<<ams-install,install status: stage 3, 100%
<<ams-install,install status: stage 4, 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. If the install process shows any error code, see Table B–1 in Appendix B, "AMS Installer Error Codes" for more information on how to resolve the error.

Once an IMlet is installed, verify it using the `ams-list` command. Here, we have added an additional IMlet: `rs232dem`. 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
```

The `ams-remove` command can be used 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 up 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
```

2

Using NetBeans or Eclipse with the Raspberry Pi Board

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

Using NetBeans with the Raspberry Pi Board

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

- NetBeans IDE 7.3 with Java ME enabled
- Oracle Java ME SDK 3.3
- Oracle Java ME SDK 3.3 NetBeans Plugin

Installing the Oracle Java ME SDK 3.3 Plugin for NetBeans

After installing NetBeans, use these steps to install the remaining software.

1. Ensure that Java ME is enabled in NetBeans. This can be done by selection **Tools** -> **Plugins** and selecting the **Installed** pane. Activate the Java ME plugin if it is not already activated.
2. Install the Java ME SDK 3.3 distribution. See the Java ME SDK 3.3 documentation for details.
3. Install the Oracle Java ME SDK 3.3 NetBeans plugin. This is a downloadable ZIP file that consists of a number of NetBeans modules (`.nbm` files) that can be added using the **Tools** -> **Plugins** dialog and selecting the **Downloaded** pane. The Oracle Java ME SDK 3.3 NetBeans plugin is required to use the Device Selector to connect to the board. See the Oracle Java ME SDK 3.3 Release Notes for installation instructions:

   `http://docs.oracle.com/javame/dev-tools/jme-sdk-3.3/release-notes/toc.htm`
4. Ensure that the Oracle Java ME Embedded 3.3 appears in the list of Java ME platforms. In the NetBeans IDE, go to **Tools** -> **Java Platforms**. If the Oracle Java ME Embedded 3.3 does not appear in the list of J2ME platforms, follow these steps:
   - Click on **Add Platform**.

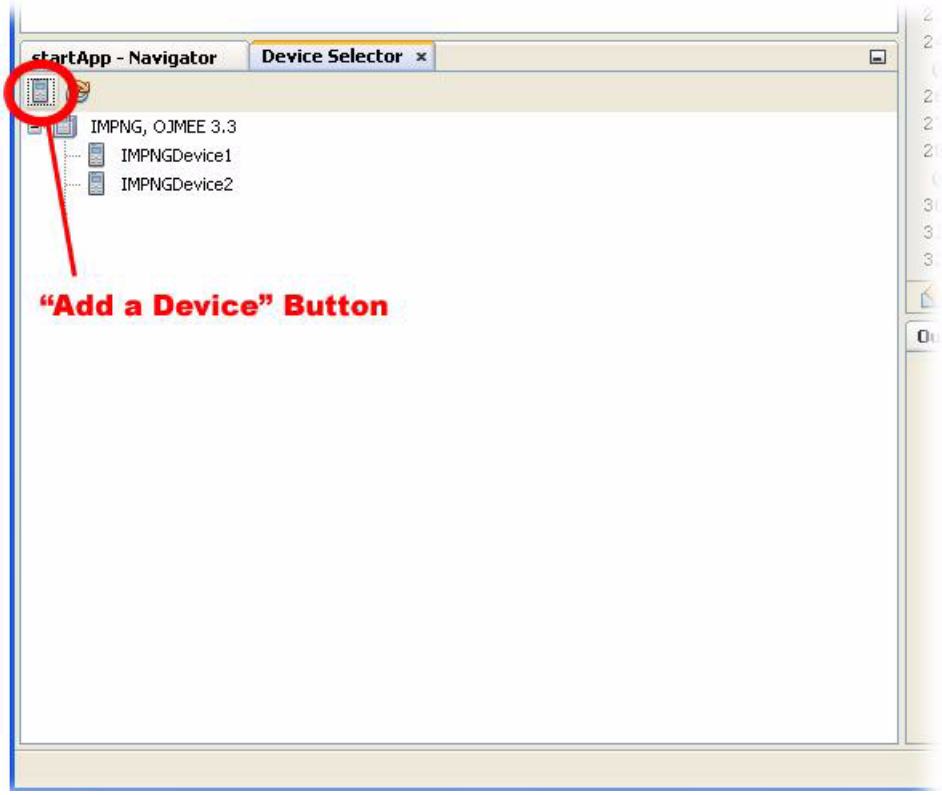
- Select **Java ME CLDC Platform Emulator** and click Next.
- Select the folder where the Oracle Java ME SDK 3.3 runtime for Raspberry Pi resides and follow the instructions to install it. Then, click **Finish** to close the dialog.

5. Ensure that the Raspberry Pi board has the Oracle Java ME Embedded distribution. See Chapter 1 for more information on how to install the runtime distribution on the Raspberry Pi board.

Adding the Raspberry Pi Board to the Device Selector

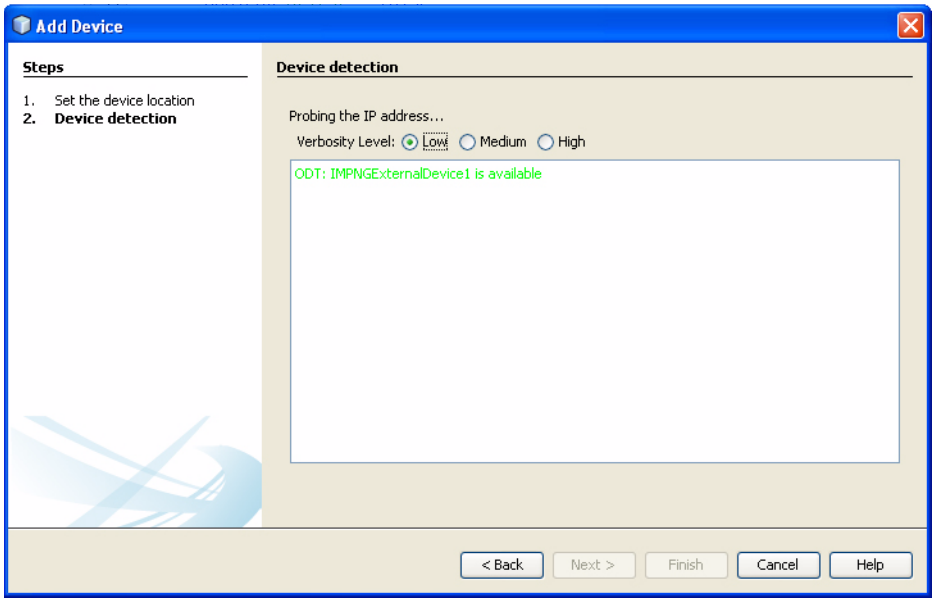
Follow these steps to add the board to the Device Selector in the Oracle Java ME SDK:

1. Ensure that the property `odt_run_on_start` is set to true in the file `bin/jwc_properties.ini` on the Raspberry Pi.
2. Ensure that the `sudo usertest.sh` script in the `/bin` directory on the Raspberry Pi is running. This allows the IDE to connect to the board. If an error occurs at any point during the debug process, you will need to restart this script.
3. If necessary, open TCP port 55123 in the firewall settings of your computer. The exact procedure to open a port differs depending on your version of Windows or the firewall software that is installed on your computer.
4. Start the NetBeans IDE. In the NetBeans IDE, go to **Tools** -> **Java ME** -> **Device Selector**
5. On the Device Selector, click on the **Add a Device** button at the top of the Device Selector window.



6. Write the IP address of the Raspberry Pi board in the **IP Address** field and click **Next**. You can find the IP address of the Raspberry Pi board by starting an

LXTerminal program and entering the `ifconfig` command. The address should be listed in the `inet addr` field in the `eth0` entry.



7. Once the device is detected, click **Finish** on the Device Detection screen.

The list of devices in the Device Selector should now include `IMPNGExternalDevice1`.

Assigning the Raspberry Pi Board to Your 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 on your project and choose **Properties**.
2. Select the **Platform** category on the properties window.
3. Select **IMPNGExternalDevice1** from the device list.

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

1. Select **File** -> **New Project**.
2. Select the **Java ME** category and **Embedded Application** in Projects. Click **Next**.
3. Provide a project name and click **Next**. Be sure that the **Create Default Package and IMlet Class** option is checked.
4. Ensure the Emulator Platform is **Oracle Java ME Embedded 3.3**. Then, select **IMPNGExternalDevice1** from the device list and click **Finish**.

After you assign the board to your project, the IMlets run on the board instead of on the emulator when you click on **Run Project** on the NetBeans IDE.

Sample Source Code

Once the project is created, use the source code in Example 2–1 for the default `IMlet.java` source file.

Example 2–1 Sample Code to Access a GPIO Port with NetBeans

```
package embeddedapplication1;
```

```
import com.oracle.deviceaccess.PeripheralManager;
import com.oracle.deviceaccess.PeripheralNotAvailableException;
import com.oracle.deviceaccess.PeripheralNotFoundException;
import com.oracle.deviceaccess.gpio.GPIOPin;
import java.io.IOException;
import javax.microedition.midlet.*;


public class IMlet extends MIDlet {

    public void startApp() {

        try {
            GPIOPin pin = (GPIOPin)PeripheralManager.open(2);
            boolean b = pin.getValue();
        } catch (PeripheralNotAvailableException ex) {
            ex.printStackTrace();
        } catch (PeripheralNotFoundException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }

    }

    public void pauseApp() {
    }

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

This sample application will obtain an object representing GPIO pin 1 from the `PeripheralManager`, and attempt to obtain its high/low value.

Accessing the Peripherals on the Raspberry Pi

There are two ways to allow access to the peripherals on the Raspberry Pi. The first is to use unsigned applications and modify the security policy file, and the second is to digitally sign the application with the appropriate API permissions requested in the JAD file.

Method #1: Modifying the Security Policy File

Modifying the security policy file is only necessary in the event that a user must manually install the application on the board, at which point the unsigned application will be installed in the `untrusted` security domain.

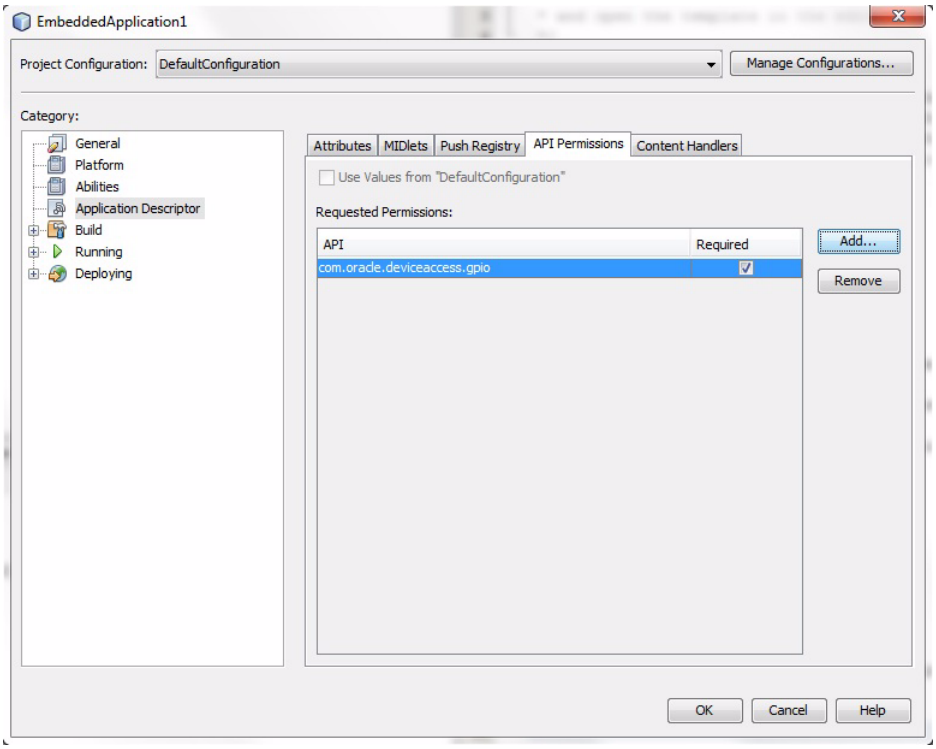
With this method, simply add the line "`allow: device_access`" to the "`untrusted`" domain of the security policy file. By default, this is located on the SD card in the `appdb/_policy.txt` file, but be sure to check the `security.policy` file entry in the `bin/jwc_properties.ini` file to verify the current file name.

Note that if an application is installed on the board using NetBeans or Eclipse during development, the application will automatically be installed in the `maximum` security domain as a convenience. Manual installation, however, will install the unsigned application into the `untrusted` security domain. Note that after development is finished, you should publish your applications with signed API permissions.

Method #2: Signing the Application with API Permissions

The second method is more complex, but is the preferred route for applications that are widely distributed. First, the JAD file must have the proper API permissions. Right-click the project name (**EmbeddedApplication1** in this example) and choose **Properties**. Select **Application Descriptor**, then in the resulting pane, select **API Permissions**. Click the **Add...** button, and add the `com.oracle.deviceaccess.gpio` API, as shown in Figure 2–1. Click **OK** to close the project properties dialog.

Figure 2–1 Adding API Permissions with NetBeans



Applications that access the Device Access APIs must also be signed. Here are the instructions on how to setup 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 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 will generate 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.

2. Copy the `appdb/_main.ks` keystore file from the Raspberry Pi over to the desktop 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 3.3 distribution.

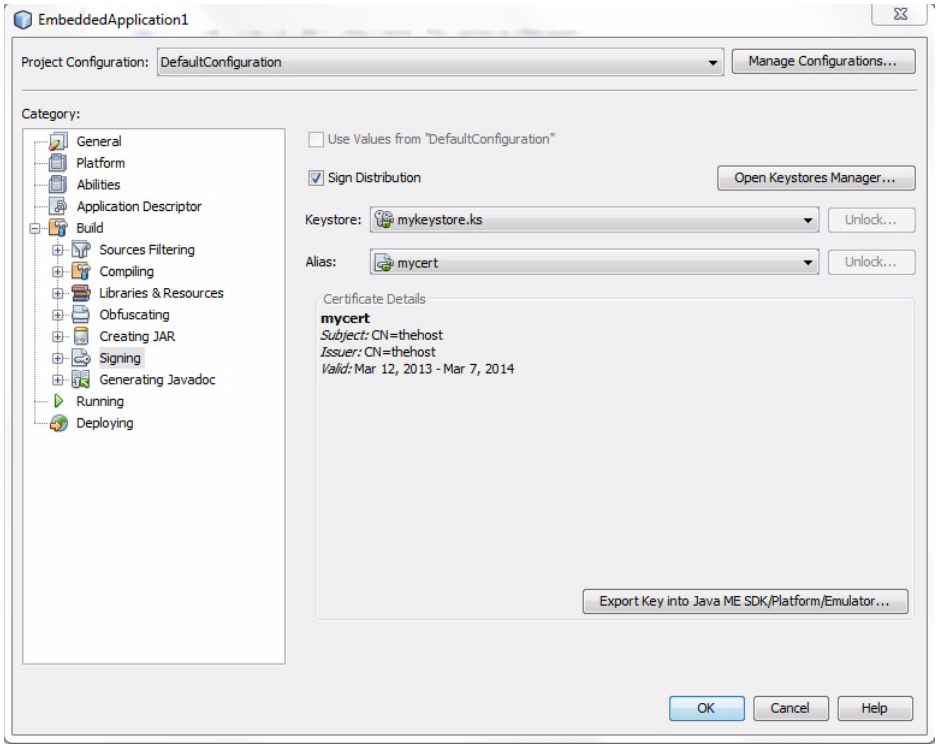
   ```
   {mekeytool} -import -MEkeystore _main.ks -keystore mykeystore.ks
   -storepass spass -alias mycert -domain trusted
   ```

This will import the information in `mykeystore.ks` you just created to the `_main.ks` keystore. Once this is completed, copy the `_main.ks` file back to its original location on the Raspberry Pi.

Use the following steps to sign your application before deploying to the Raspberry Pi board.

1. Right click your project and select **Properties**.
2. Choose the **Signing** option under the **Build** category.
3. Open the **Keystores Manager** and import the `mykeystore.ks` file that you created.
4. Check the **Sign Distribution** box. If you wish, unlock the keystore and the key with the passwords that you specified earlier. This is shown in Figure 2–2.
5. When the project is built and run, it will be digitally signed and deployed to the Raspberry Pi.

Figure 2–2 The Signing Pane in the NetBeans Project Properties



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** -> **Debug Project** or use the Debug button on the toolbar.

The debugger connects to the debug agent on the board and the program execution stops at your breakpoint, as shown in Figure 2–3.

Figure 2–3 Debugging an IMlet on the Board Using NetBeans

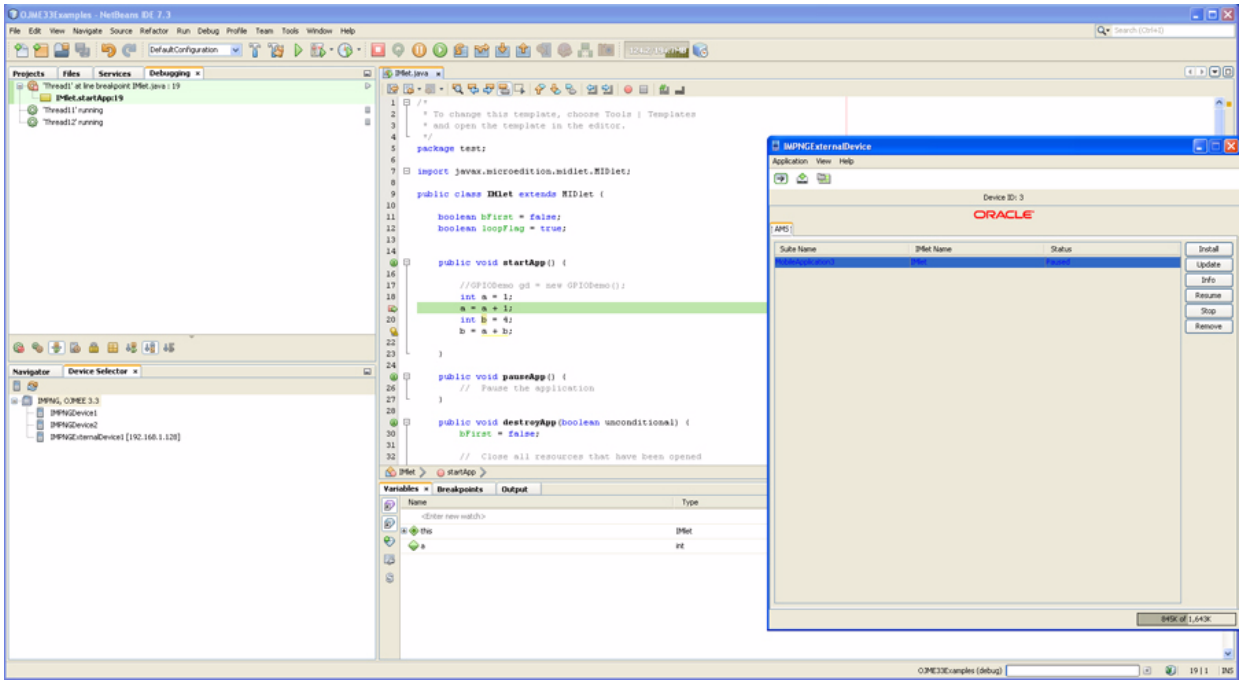


Figure 2–3 shown an entire NetBeans debugging environment that allows the programmer to execute a program step by step as well as add and remove variables from a watch list on the bottom of the screen.

For more information on using the device access APIs, please see the Device Access API Guide and the associated javadocs.

Using Eclipse with the Raspberry Pi Board

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

- Eclipse 3.7 Indigo or Eclipse 4.2 Juno
- Oracle Java ME SDK 3.3
- Oracle Java ME SDK 3.3 Eclipse Plugin

Installing the Oracle Java ME SDK 3.3 Plugin for Eclipse

After installing Eclipse, use these steps to install the remaining software.

1. Install the Java ME SDK 3.3 distribution. See the Java ME SDK 3.3 documentation for details.
2. Install the Oracle Java ME SDK 3.3 Eclipse plugin. This is required to use the Device Selector to connect to the board. See the Oracle Java ME SDK 3.3 Release Notes for installation instructions:

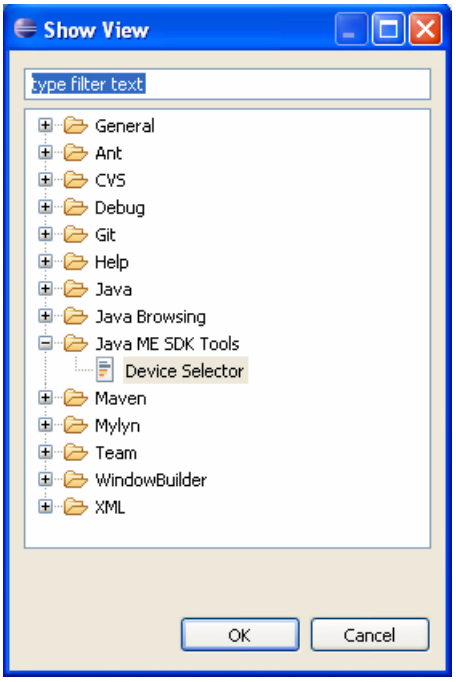
   `http://docs.oracle.com/javame/dev-tools/jme-sdk-3.3/release-notes/toc.htm`
3. Ensure that the Raspberry Pi board has the Oracle Java ME Embedded 3.3 runtime. See Chapter 1 for more information on how to install the runtime distribution on the Raspberry Pi board.

4. Ensure that the Oracle Java ME Embedded 3.3 appears in the list of Java ME platforms. If it doesn't appear, follow these steps for your project properties.
   - Under the **Java ME** category, select **Device Management**. In the Device Management window, press the **Manual Install...** button.
   - The Manual Device Installation window appears, without the Oracle Java ME Embedded devices. Press the **Browse** button. A browser window appears.
   - Browse to the base directory of the Java ME SDK environment and press the **OK** button. After the platform is scanned and the devices are installed, close each of the respective dialogs.

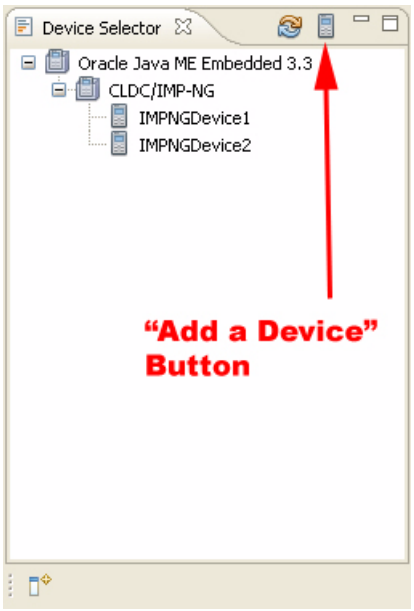
Adding the Raspberry Pi Board to the Device Selector

Follow these steps to add the board to the Device Selector in the Oracle Java ME SDK:

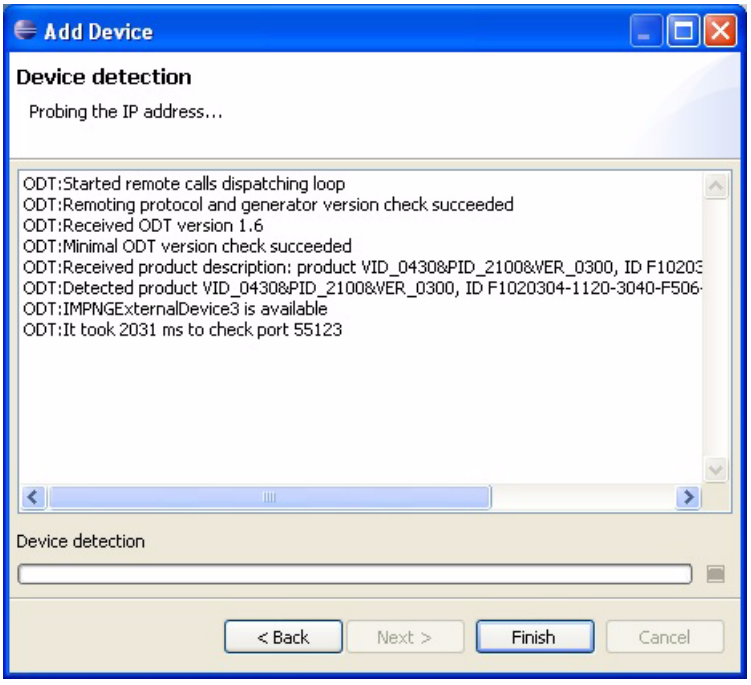
1. Ensure that the property `odt_run_on_start` is set to true in the file `bin/jwc_properties.ini` on the Raspberry Pi.
2. Ensure that the `sudo usertest.sh` script in the `/bin` directory on the Raspberry Pi is running. This allows the IDE to connect to the board. If an error occurs at any point during the debug process, you will need to restart this script.
3. If necessary, open TCP port 55123 in the firewall settings of your computer. The exact procedure to open a port differs depending on your version of Windows or the firewall software that is installed on your computer.
4. Start the Eclipse IDE. In the Eclipse IDE, go to **Window** -> **Show View** -> **Other**. In the popup window that appears, expand the **Java ME** node and select **Device Selector**.



5. On the Device Selector, click on the **Add a Device** button at the top of the Device Selector window.



6. Write the IP address of the Raspberry Pi board in the **IP Address** field and click **Next**. You can find the IP address of the Raspberry Pi board by starting an LXTerminal program and entering the `ifconfig` command. The address should be listed in the `inet addr` field in the `eth0` entry.



7. Once the device is detected, click **Finish** on the Add Device screen.

The list of devices in the Device Selector should now include `IMPNGExternalDevice1`.

Assigning the Raspberry Pi Board to Your Project

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

1. Right-click on your project and choose **Properties**.
2. Select the **Java ME** category on the properties window.
3. Select **IMPNGExternalDevice1** from the device list. If the device is not shown, add it using the **Add...** button, selecting **Oracle Java ME Embedded 3.3** as the SDK and **IMPNGExternalDevice1** as the device.

If you are creating a new Eclipse project from scratch, follow these steps:

1. Select **New** -> **Other**. Then expand the **Java ME** tree node, and create a new **MIDlet Project**.
2. Expand the **Java ME** tree node, and create a new **MIDlet Project**.
3. In the Configuration pane of the creation dialog, select **IMPNGExternalDevice1** from the device list.
4. Select the appropriate **Profile** and **Configuration** for your project.

After you assign the board to your project, the IMlets run on the board instead of on the emulator when you click on **Project** -> **Run** on the Eclipse IDE.

Sample Source Code

Once the project is created, use the source code in Example 2–2 for a default source file.

Example 2–2 Sample Code to Access a GPIO Port with Eclipse

```
package embeddedapplication1;

import com.oracle.deviceaccess.PeripheralManager;
import com.oracle.deviceaccess.PeripheralNotAvailableException;
import com.oracle.deviceaccess.PeripheralNotFoundException;
import com.oracle.deviceaccess.gpio.GPIOPin;
import java.io.IOException;
import javax.microedition.midlet.*;


public class IMlet extends MIDlet {

    public void startApp() {

        try {
            GPIOPin pin = (GPIOPin)PeripheralManager.open(2);
            boolean b = pin.getValue();
        } catch (PeripheralNotAvailableException ex) {
            ex.printStackTrace();
        } catch (PeripheralNotFoundException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }

    }

    public void pauseApp() {
    }
```

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

This sample application will obtain an object representing GPIO pin 1 from the `PeripheralManager`, and attempt to obtain its high/low value.

Accessing the Peripherals on the Raspberry Pi

There are two ways to allow access to the peripherals on the Raspberry Pi. The first is to use unsigned applications and modify the security policy file, and the second is to digitally sign the application with the appropriate API permissions requested in the JAD file.

Method #1: Modifying the Security Policy File

Modifying the security policy file is only necessary in the event that a user must manually install the application on the board, at which point the unsigned application will be installed in the `untrusted` security domain.

With this method, simply add the line "`allow: device_access`" to the "`untrusted`" domain of the security policy file. By default, this is located on the SD card in the `appdb/_policy.txt` file, but be sure to check the `security.policy` file entry in the `bin/jwc_properties.ini` file to verify the current file name.

Note that if an application is installed on the board using NetBeans or Eclipse during development, the application will automatically be installed in the `maximum` security domain as a convenience. Manual installation, however, will install the unsigned application into the `untrusted` security domain. Note that after development is finished, you should publish your applications with signed API permissions.

Method #2: Signing the Application with API Permissions

The second method is more complex, but is the preferred route for applications that are widely distributed. Open the Application Descriptor for your project in the Packages window, and select the **Application Descriptor** pane. You will need to manually add or change the following lines in the Application Descriptor.

```
MIDlet-Permissions: com.oracle.deviceaccess.gpio
Microedition-Profile: IMP-NG
```

Applications that access the Device Access APIs must also be signed. Here are the instructions on how to setup 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 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 will generate 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.

2. Copy the `appdb/_main.ks` keystore file from the Raspberry Pi over to the desktop 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 3.3 distribution.

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

This will import the information in `mykeystore.ks` you just created to the `_main.ks` keystore. Once this is completed, copy the `_main.ks` file back to its original location on the Raspberry Pi.

Use the following steps to sign your application before deploying to the Raspberry Pi board.

1. Right click your project and select **Properties**.
2. Choose the **Signing** option under the **Java ME** category.
3. Check the **Enable Project Specific Settings** checkbox. Import the `mykeystore.ks` file that you created as an **External...** keystore file. Provide the keystore and key passwords that you created earlier. Ensure that the **mycert** key alias is present.
4. Ensure that the project is being signed in the project's Application Descriptor. When the project is built and run, it will be digitally signed when deployed to the Raspberry Pi.

Debugging an IMlet on the Raspberry Pi Board

After you assign the board to your project, follow these steps to debug an IMlet:

1. Open your IMlet class on the Eclipse editor.
2. Click once directly on the line number where you want to set a breakpoint. The line number has a small circle next to it to indicate a breakpoint.
3. Select **Run** -> **Debug** or use the Debug button on the toolbar.

The debugger connects to the debug agent on the board and the program execution stops at your breakpoint, as shown in Figure 2–4.

Figure 2–4 Debugging an IMlet on the Board Using the Eclipse IDE

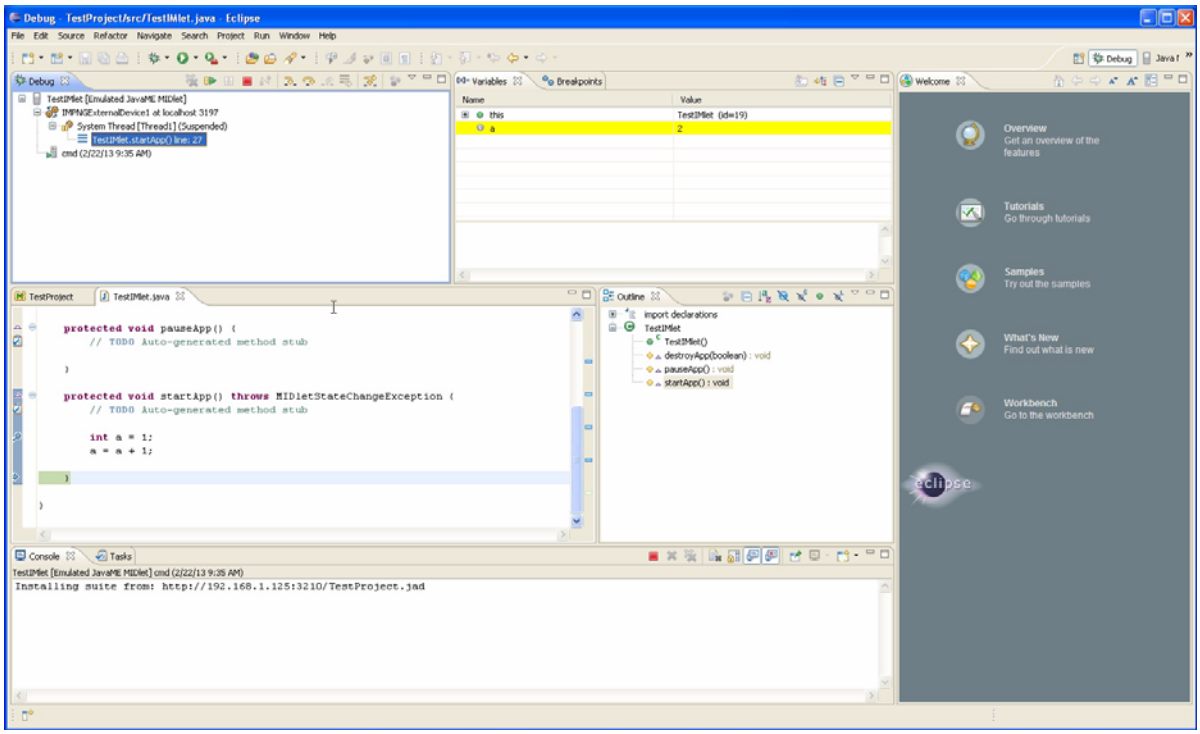


Figure 2–4 shown an entire Eclipse debugging environment that allows the programmer to execute a program step by step as well as add and remove variables from a watch list on the bottom of the screen.

For more information on using the device access APIs, please see the Device Access API Guide and the associated javadocs.

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 four categories:

- Installing Linux on the Raspberry Pi Board
- Starting Oracle Java ME Embedded 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 cannot find a valid disk image on the SD card.	Be sure to 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 is missing or corrupted.	Replace the following files: ■ 3 flashes: `loader.bin` not found ■ 4 flashes: `loader.bin` not launched ■ 5 flashes: `start.elf` not found ■ 6 flashes:`start.elf` not launched ■ 7 flashes: `kernel.img` not found

Starting Oracle Java ME Embedded 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 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.	Check 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 Pi does not have the startup sequence activated.	Use the Raspberry Pi setup application to set the desktop to activate at boot.

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.	On-device debugging is not enabled.	Edit the file `jwc_properties.ini` and set the property `odt_run_on_start` to `true`.
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.

A

Raspberry Pi Board Peripheral 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 Access APIs. Note that any IMlet that accesses the Device Access 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 Access APIs.

GPIO Pins

The following GPIO pins are pre-configured.

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
2	GPIO2	GPIO 2	portNumber = 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	portNumber = 0 pinNumber = 3 direction = GPIOPinConfig.DIR_INPUT_ONLY mode = GPIOPinConfig.MODE_INPUT_PULL_UP trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES initValue - ignored

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
4	GPIO4	GPIO 4	portNumber = 0 pinNumber = 4 direction = GPIOPinConfig.DIR_INPUT_ONLY mode = PeripheralConfig.DEFAULT trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES initValue - ignored
7	GPIO7	GPIO 7	portNumber = 0 pinNumber = 7 direction = GPIOPinConfig.DIR_OUTPUT_ONLY mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES initValue = false
8	GPIO8	GPIO 8	portNumber = 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	portNumber = 0 pinNumber = 9 direction = GPIOPinConfig.DIR_INPUT_ONLY mode = PeripheralConfig.DEFAULT trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES initValue - ignored
10	GPIO10	GPIO 10	portNumber = 0 pinNumber = 10 direction = GPIOPinConfig.DIR_INPUT_ONLY mode = PeripheralConfig.DEFAULT trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES initValue - ignored

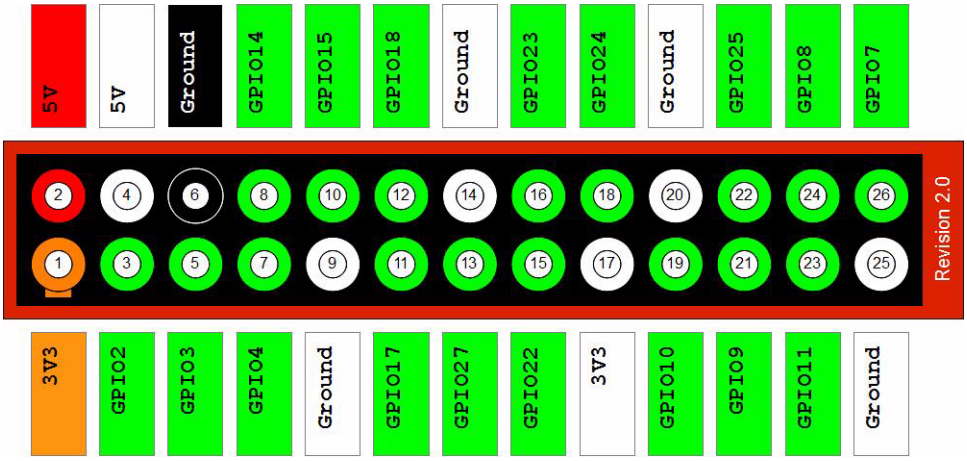
DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
11	GPIO11	GPIO 11	portNumber = 0 pinNumber = 11 direction = GPIOPinConfig.DIR_INPUT_ONLY mode = PeripheralConfig.DEFAULT trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES initValue - ignored
14	GPIO14	GPIO 14	portNumber = 0 pinNumber = 14 direction = GPIOPinConfig.DIR_OUTPUT_ONLY mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES initValue = false
15	GPIO15	GPIO 15	portNumber = 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	portNumber = 0 pinNumber = 17 direction = GPIOPinConfig.DIR_INPUT_ONLY mode = PeripheralConfig.DEFAULT trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES initValue - ignored
18	GPIO18	GPIO 18	portNumber = 0 pinNumber = 18 direction = GPIOPinConfig.DIR_OUTPUT_ONLY mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL trigger - ignored initValue = false

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
22	GPIO22	GPIO 22	portNumber = 0 pinNumber = 22 direction = GPIOPinConfig.DIR_INPUT_ONLY mode = PeripheralConfig.DEFAULT trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES initValue - ignored
23	GPIO23	GPIO 23	portNumber = 0 pinNumber = 23 direction = GPIOPinConfig.DIR_OUTPUT_ONLY mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL trigger - ignored initValue = false
24	GPIO24	GPIO 24	portNumber = 0 pinNumber = 24 direction = GPIOPinConfig.DIR_OUTPUT_ONLY mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL trigger - ignored initValue = false
25	GPIO25	GPIO 25	portNumber = 0 pinNumber = 25 direction = GPIOPinConfig.DIR_OUTPUT_ONLY mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL trigger - ignored initValue = false
27	GPIO27	GPIO 27	portNumber = 0 pinNumber = 27 direction = GPIOPinConfig.DIR_INPUT_ONLY mode = PeripheralConfig.DEFAULT trigger = GPIOPinConfig.TRIGGER_BOTH_EDGES initValue - ignored

Please note the following items concering GPIO on the Raspberry Pi.

- The value of `PeripheralConfig.DEFAULT` when applied to the `portNumber` is 0.

- The value of `PeripheralConfig.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 doesn't allow any communication with a loopback device. The following configuration, however, can be used to communicate to I2C slaves.

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
NONE		GPIO 2 (SDA) GPIO 3 (SCL)	

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

- 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). For example, Example to configure a TCS3414-A I2C color sensor on a revision 2 Raspberry Pi board, use the following constructor: `I2CDeviceConfig(1, 57, 7, 100000)`
- The value of `PeripheralConfig.DEFAULT` when applied to the `busNumber` is 0.

- The value of `PeripheralConfig.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` (this is probably commented out in the file `/etc/modprobe.d/raspi-blacklist.conf`; simply uncomment it and reboot the device to apply the changes) and `i2c-dev` (you can add `i2c-dev`, without quotes, to the `/etc/modules` file and reboot to apply the changes).
- 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:

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
31	PWM		byteOrdering = Peripheral.LITTLE_ ENDIAN

The MMIO raw memories are shown here:

DAAPI Peripheral ID	Name	Address	Type and Size
31	CTL	0x7e20C000	int 4
31	STA	0x7e20C004	int 4
31	RNG1	0x7e20C010	int 4
31	DAT1	0x7e20C014	int 4
31	FIF1	0x7e20C018	int 4

There are no devices with event support. Due to nature of memory organization of the Raspberry Pi, programmers can create a custom `MMIODeviceConfig` to access the following memory ranges. Note that all are IO Peripheral register ranges with exclusion of DMA regsiters. The end addresses are not inclusive.

- {0x7E215000, 0x7E2150D8},
- {0x7E205000, 0x7E20501f},
- {0x7E804000, 0x7E80401f},
- {0x7E805000, 0x7E80501f},
- {0x7E300000, 0x7E3000ff},
- {0x7E200000, 0x7E2000B4},
- {0x7E203000, 0x7E203024},
- {0x7e20C000, 0x7e20C028},

- {0x7E204000, 0x7E204018},
- {0x7E214000, 0x7E21403f},
- {0x7E003000, 0x7E00301b},
- {0x7E201000, 0x7E20108f},
- {0x7E00B400, 0x7E00B424}

Only `int` types for the memory configuration are allowed. Otherwise, an `IOException` will be thrown.

SPI

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

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	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 `PeripheralConfig.DEFAULT` when applied to the `busNumber` is 0.
- The value of `PeripheralConfig.DEFAULT` when applied to the `clockFrequency` is 2000000 Hz.
- The value of `PeripheralConfig.DEFAULT` when applied to the `wordLength` is 8.
- The value of `PeripheralConfig.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 `raspi-blacklist.conf` file and reboot the board.
- Only 8-bit word lengths are supported on the Raspberry Pi.

UART

The following UART devices are pre-configured:

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
40	UART	GPIO 14 (TXD) GPIO 15 (RXD)	uartNumber = 0 (see below) 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 environement, 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:

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
30	WDG	Platform Watchdog	

B

AMS Installer Error Codes

Table B–1 lists the error codes that the AMS command-line interface shows when the installation of an IMlet fails. The description of each code contains more information about the problem that caused the error.

Table B–1 Installer Error Codes

Constant	Error Code	Description
ALAA_ALIAS_NOT_FOUND	78	Application Level Access Authorization: The alias definition is missing.
ALAA_ALIAS_WRONG	80	Application Level Access Authorization: The alias definition is wrong.
ALAA_MULTIPLE_ALIAS	79	Application Level Access Authorization: An alias has multiple entries that match.
ALAA_TYPE_WRONG	77	Application Level Access Authorization: The `MIDlet-Access-Auth-Type` has missing parameters.
ALREADY_INSTALLED	39	The JAD matches a version of a suite already installed.
APP_INTEGRITY_FAILURE_ DEPENDENCY_CONFLICT	69	Application Integrity Failure: two or more dependencies exist on the component with the same name and vendor, but have different versions or hashs.
APP_INTEGRITY_FAILURE_ DEPENDENCY_MISMATCH	70	Application Integrity Failure: there is a component name or vendor mismatch between the component JAD and IMlet or component JAD that depends on it.
APP_INTEGRITY_FAILURE_HASH_ MISMATCH	68	Application Integrity Failure: hash mismatch.
ATTRIBUTE_MISMATCH	50	A attribute in both the JAD and JAR manifest does not match.
AUTHORIZATION_FAILURE	49	Application authorization failure, possibly indicating that the application was not digitally signed.
CA_DISABLED	60	Indicates that the trusted certificate authority (CA) for this suite has been disabled for software authorization.
CANCELED	101	Canceled by user.
CANNOT_AUTH	35	The server does not support basic authentication.

Table B–1 (Cont.) Installer Error Codes

Constant	Error Code	Description
CIRCULAR_COMPONENT_DEPENDENCY	64	Circular dynamic component dependency.
COMPONENT_DEPS_LIMIT_EXCEEDED	65	Dynamic component dependencies limit exceeded.
COMPONENT_NAMESPACE_COLLISION	72	The namespace used by a component is the same as another.
CONTENT_HANDLER_CONFLICT	55	The installation of a content handler would conflict with an already installed handler.
CORRUPT_DEPENDENCY_HASH	71	A dependency has a corrupt hash code.
CORRUPT_JAR	36	An entry could not be read from the JAR.
CORRUPT_PROVIDER_CERT	5	The content provider certificate cannot be decoded.
CORRUPT_SIGNATURE	8	The JAR signature cannot be decoded.
DEVICE_INCOMPATIBLE	40	The device does not support either the configuration or profile in the JAD.
DUPLICATED_KEY	88	Duplicated JAD/manifest key attribute
EXPIRED_CA_KEY	12	The certificate authority's public key has expired.
EXPIRED_PROVIDER_CERT	11	The content provider certificate has expired.
INCORRECT_FONT_LOADING	82	A font that is contained with the JAR cannot be loaded.
INSUFFICIENT_STORAGE	30	Not enough storage for this suite to be installed.
INVALID_CONTENT_HANDLER	54	The `MicroEdition-Handler-<n>` JAD attribute has invalid values.
INVALID_JAD_TYPE	37	The server did not have a resource with the correct type or the JAD downloaded has the wrong media type.
INVALID_JAD_URL	43	The JAD URL is invalid.
INVALID_JAR_TYPE	38	The server did not have a resource with the correct type or the JAR downloaded has the wrong media type.
INVALID_JAR_URL	44	The JAR URL is invalid.
INVALID_KEY	28	A key for an attribute is not formatted correctly.
INVALID_NATIVE_LIBRARY	85	A native library contained within the JAR cannot be loaded.
INVALID_PACKAGING	87	A dependency cannot be satisfied.
INVALID_PAYMENT_INFO	58	Indicates that the payment information provided with the IMlet suite is incomplete or incorrect.
INVALID_PROVIDER_CERT	7	The signature of the content provider certificate is invalid.

Table B–1 (Cont.) Installer Error Codes

Constant	Error Code	Description
INVALID_RMS_DATA_TYPE	76	The server did not have a resource with the correct type or the JAD downloaded has the wrong media type.
INVALID_RMS_DATA_URL	73	The RMS data file URL is invalid.
INVALID_SERVICE_EXPORT	86	A LIBlet that exports a service with a LIBlet Services attribute does not contain the matching service provider configuration information.
INVALID_SIGNATURE	9	The signature of the JAR is invalid.
INVALID_VALUE	29	A value for an attribute is not formatted correctly.
INVALID_VERSION	16	The format of the version is invalid.
IO_ERROR	102	A low-level hardware error has occurred.
JAD_MOVED	34	The JAD URL for an installed suite is different than the original JAD URL.
JAD_NOT_FOUND	2	The JAD was not found.
JAD_SERVER_NOT_FOUND	1	The server for the JAD was not found.
JAR_CLASSES_VERIFICATION_FAILED	56	Not all classes within JAR package can be successfully verified with class verifier.
JAR_IS_LOCKED	100	Component or MIDlet or IMlet suite is locked by the system.
JAR_NOT_FOUND	20	The JAR was not found at the URL given in the JAD.
JAR_SERVER_NOT_FOUND	19	The server for the JAR was not found at the URL given in the JAD.
JAR_SIZE_MISMATCH	31	The JAR downloaded was not the same size as given in the JAD.
MISSING_CONFIGURATION	41	The configuration is missing from the manifest.
MISSING_DEPENDENCY_HASH	67	A dependency hash code is missing.
MISSING_DEPENDENCY_JAD_URL	66	A dependency JAD URL is missing.
MISSING_JAR_SIZE	21	The JAR size is missing.
MISSING_JAR_URL	18	The URL for the JAR is missing.
MISSING_PROFILE	42	The profile is missing from the manifest.
MISSING_PROVIDER_CERT	4	The content provider certificate is missing.
MISSING_SUITE_NAME	13	The name of MIDlet or IMlet suite is missing.
MISSING_VENDOR	14	The vendor is missing.
MISSING_VERSION	15	The version is missing.
NEW_VERSION	32	This suite is newer that the one currently installed.

Table B–1 (Cont.) Installer Error Codes

Constant	Error Code	Description
NO_ERROR	0	No error.
NOT_YET_VALID_PROVIDER_CERT	89	A certificate is not yet valid.
NOT_YET_VALID_CA_KEY	90	A CA's public key is not yet valid.
OLD_VERSION	17	This suite is older that the one currently installed.
OTHER_ERROR	103	Other errors.
PROXY_AUTH	51	Indicates that the user must first authenticate with the proxy.
PUSH_CLASS_FAILURE	48	The class in a push attribute is not in MIDlet-<*n*> attribute.
PUSH_DUP_FAILURE	45	The connection in a push entry is already taken.
PUSH_FORMAT_FAILURE	46	The format of a push attribute has an invalid format.
PUSH_PROTO_FAILURE	47	The connection in a push attribute is not supported.
REVOKED_CERT	62	The certificate has been revoked.
RMS_DATA_DECRYPT_PASSWORD	83	Indicates that a password is required to decrypt RMS data.
RMS_DATA_ENCRYPT_PASSWORD	84	Indicates that a password is required to encrypt RMS data.
RMS_DATA_NOT_FOUND	75	The RMS data file was not found at the specified URL.
RMS_DATA_SERVER_NOT_FOUND	74	The server for the RMS data file was not found at the specified URL.
RMS_INITIALIZATION_FAILURE	81	Failure to import RMS data.
SUITE_NAME_MISMATCH	25	The MIDlet or IMlet suite name does not match the one in the JAR manifest.
TOO_MANY_PROPS	53	Indicates that either the JAD or manifest has too many properties to fit into memory.
TRUSTED_OVERWRITE_FAILURE	52	Indicates that the user tried to overwrite a trusted suite with an untrusted suite during an update.
UNAUTHORIZED	33	Web server authentication failed or is required.
UNKNOWN_CA	6	The certificate authority (CA) that issued the content provider certificate is unknown.
UNKNOWN_CERT_STATUS	63	The certificate is unknown to OCSP server.
UNSUPPORTED_CERT	10	The content provider certificate has an unsupported version.
UNSUPPORTED_CHAR_ENCODING	61	Indicates that the character encoding specified in the MIME type is not supported.

Table B–1 (Cont.) Installer Error Codes

Constant	Error Code	Description
UNSUPPORTED_PAYMENT_INFO	57	Indicates that the payment information provided with the MIDlet or IMlet suite is incompatible with the current implementation.
UNTRUSTED_PAYMENT_SUITE	59	Indicates that the MIDlet or IMlet suite has payment provisioning information but it is not trusted.
VENDOR_MISMATCH	27	The vendor does not match the one in the JAR manifest.
VERSION_MISMATCH	26	The version does not match the one in the JAR manifest.

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) and is 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 part of CLDC, it is 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 Input/Output. 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 Socket Layer (SSL) technology.

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 can not 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.

I2C

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

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.

JCP

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

JDTS

Java Device Test Suite. A set of Java programming language tests developed specifically for the wireless marketplace, providing targeted, standardized testing for CLDC and MIDP on small and handheld devices.

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.

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.

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.

Preemption

Taking a resource, such as the foreground, from another application.

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 non-volatile 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. The SMSC routes messages and regulates **SMS** traffic. When an SMS message is sent, it goes to an SMS center first, 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 contact 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, 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.

Index

A

P