Oracle® Java ME Embedded

Getting Started Guide for the Reference Platform (Qualcomm IoE)

Release 8

E48513-03

July 2014

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

Oracle Java ME Embedded Getting Started Guide for the Reference Platform (Qualcomm IoE), Release 8

E48513-03

Contents

List of Figures

List of Tables

Preface

This guide describes how to install and configure Oracle Java ME Embedded software onto a Qualcomm Internet-of-Everything (IoE) embedded device. In addition, it contains troubleshooting information and Device I/O API specifications useful for Java developers.

Audience

This guide is for developers who want to run Oracle Java ME Embedded software on a Qualcomm IoE 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

This guide frequently references the *Qualcomm IoE Development Platform User Guide*, which can be downloaded at:

`https://developer.qualcomm.com/mobile-development/development-devices-boards/development-boards/internet-of-everything-development-platform/tools-and-resources`

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

Shell Prompts

Shell	Prompt
Windows	*directory*>

Shell	Prompt
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 Qualcomm IoE Board

This chapter demonstrates how to install the Oracle Java ME Embedded software on the Qualcomm Internet-of-Everything (IoE) board. The following items are required for installing and developing on the Qualcomm IoE board:

- Qualcomm IoE board running the Brew MP operating system
- Desktop PC running Microsoft Windows 7
- USB cable with a micro-B connector that can link the Qualcomm IoE board to your desktop PC
- Qualcomm IoE USB drivers
- Qualcomm Brew MP SDK tools
- Terminal emulator program, such as PuTTY.
- Oracle Java ME Embedded 8
- Oracle Java ME SDK 8
- NetBeans IDE 8.0

Setting Up the Qualcomm IoE Board

Download the *Qualcomm IoE Development Platform User Guide* in PDF format from:

`https://developer.qualcomm.com/mobile-development/development-devices-boards/development-boards/internet-of-everything-development-platform/`

This guide contains important information about the Qualcomm IoE board and its hardware.

Perform the following steps.

1. Assemble and connect the board's components as listed in Chapter 3, "IoE Development Platform Hardware," and Chapter 5, "Hardware Configuration," of the *Qualcomm IoE Development Platform User Guide*.
2. Connect the power supply as shown in the *Qualcomm IoE Development Platform User Guide* and power on the board. Follow the instructions in these sections:
    - To use an AC power source, see Section 5.3.5.5, "AC-powered operation."
    - To use a battery source, see Section 5.3.5.4, "Battery-powered operation."

Installing the Qualcomm IoE USB Drivers

To develop on the Qualcomm IoE board, you must first install the Windows USB drivers for the board. To install the drivers, follow these steps:

1. Download the Qualcomm IoE USB drivers from the following site:

   `https://developer.qualcomm.com/mobile-development/development-devices-boards/development-boards/internet-of-everything-development-platform/tools-and-resources`

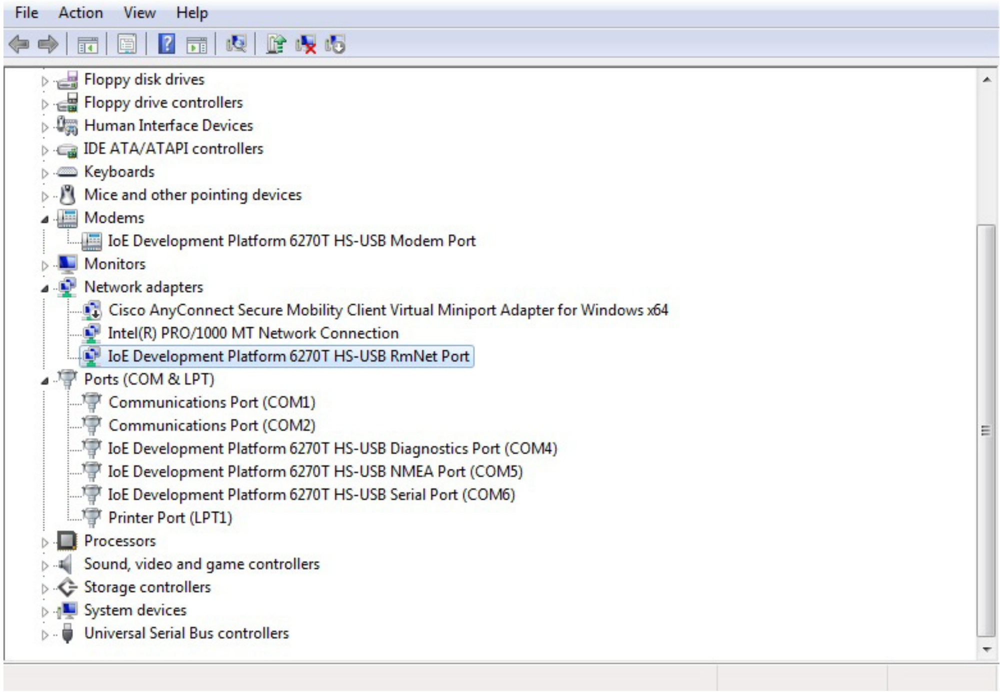
2. Ensure that the Qualcomm IoE board is powered on by pressing the PWR KEY button on the board. Then, follow the instructions in Chapter 2, "Software Setup," of the *Qualcomm IoE Development Platform User Guide* to properly install the USB drivers on Windows.

3. Open the Windows Device Manager (by clicking **Start** then **Device Manager**).

4. Ensure that the drivers are successfully installed by verifying the following hardware ports, as shown in Figure 1–1:

   - HS-USB modem port
   - HS-USB serial port
   - HS-USB diagnostics port
   - HS-USB NMEA port
   - HS-USB RmNet port (optional and reserved)

Figure 1–1 Device Manager with Qualcomm IoE USB Device Drivers Loaded



- Floppy disk drives
- Floppy drive controllers
- Human Interface Devices
- IDE ATA/ATAPI controllers
- Keyboards
- Mice and other pointing devices
- Modems
  - IoE Development Platform 6270T HS-USB Modem Port
- Monitors
- Network adapters
  - Cisco AnyConnect Secure Mobility Client Virtual Miniport Adapter for Windows x64
  - Intel(R) PRO/1000 MT Network Connection
  - IoE Development Platform 6270T HS-USB RmNet Port
- Ports (COM & LPT)
  - Communications Port (COM1)
  - Communications Port (COM2)
  - IoE Development Platform 6270T HS-USB Diagnostics Port (COM4)
  - IoE Development Platform 6270T HS-USB NMEA Port (COM5)
  - IoE Development Platform 6270T HS-USB Serial Port (COM6)
  - Printer Port (LPT1)
- Processors
- Sound, video and game controllers
- Storage controllers
- System devices
- Universal Serial Bus controllers

Downloading and Installing the Brew MP SDK Tools

Download the Brew MP SDK tools from the following site:
`https://developer.brewmp.com/tools/brew-mp-sdk`

The version used in this documentation is 7.12.5. Double-click the installer executable file, and install the application on your Windows platform desktop as per the instructions. You will need the **Loader** and **Logger** applications that are installed with the Brew MP SDK Tools in later sections.

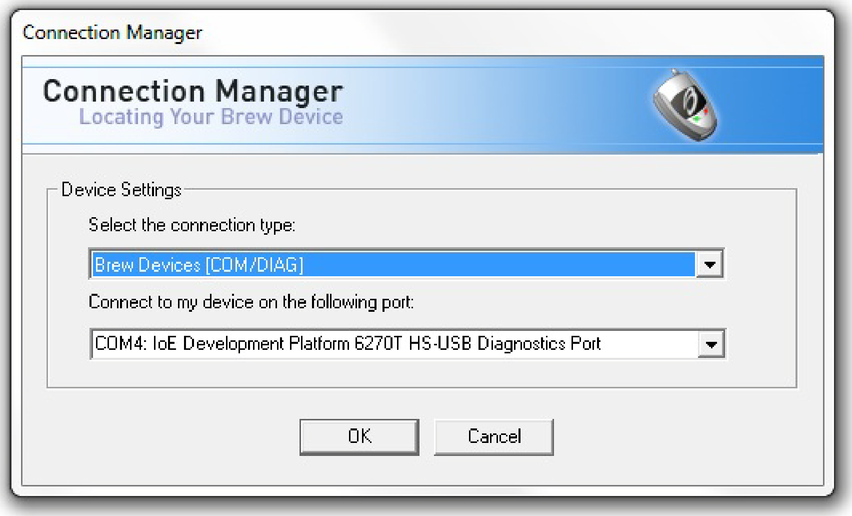
Copying Files to the Qualcomm IoE Board

You must install the Oracle Java ME Embedded software on the Qualcomm IoE board using the **Loader** and **Logger** applications. To copy the appropriate files to the board:

1. Download and uncompress Oracle Java ME Embedded 8 for the Qualcomm IoE board.
2. Get the latest `java.sig` signature file from the Qualcomm IoE website:

   `https://developer.qualcomm.com/mobile-development/development-devices-boards/development-boards/internet-of-everything-development-platform/tools-and-resources`
3. Start the **Loader** application that was installed with the Brew MP SDK.
4. When the **Loader** application starts, connect to the board using the Connection type: **Brew Devices (COM/DIAG)**, and whichever port matches the Qualcomm HS-USB Diagnostics Port, as shown in Figure 1–2.

Figure 1–2 Brew SDK Loader Connection Manager



Connection Manager

Connection Manager
Locating Your Brew Device

Device Settings

Select the connection type: Brew Devices [COM/DIAG]

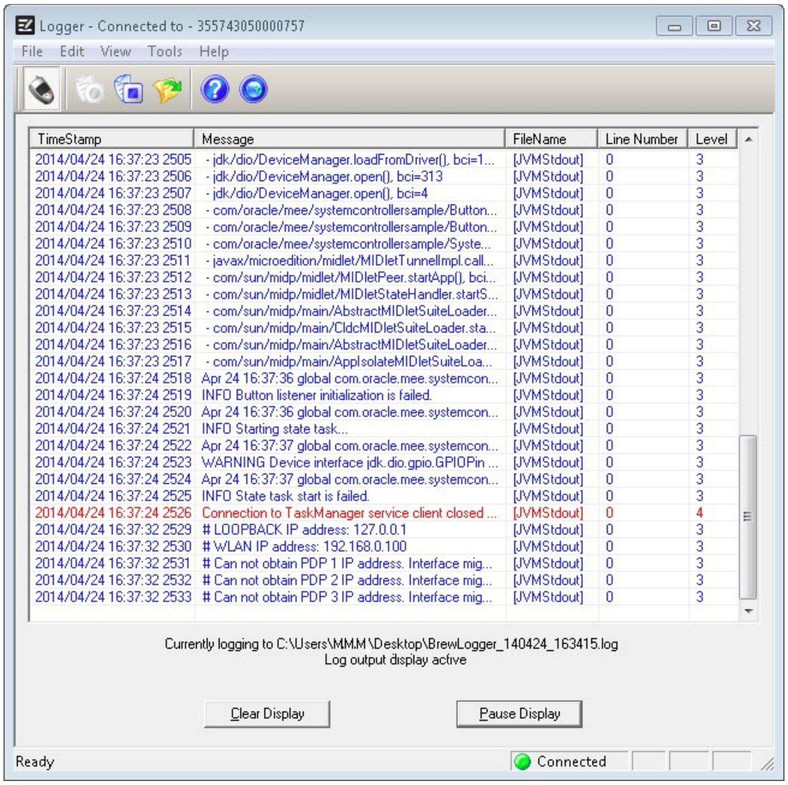
Connect to my device on the following port: COM4: IoE Development Platform 6270T HS-USB Diagnostics Port

5. If you are upgrading from an earlier release, such as Oracle Java ME Embedded 3.4, or Oracle Java ME Embedded 8 EA or EA2, then you must delete all the files in the `/sys/mod/java` and `/sys/mod/netsetup` directories, including the`.sig` file. After deleting the files, reboot the board.
6. If it does not exist, then create the `/sys/mod/java` directory and drag and drop the following files from the Oracle Java ME Embedded distribution's `java` folder into that directory.

   - `appdb` (folder)
   - `java.mif`
   - `java.mod`
   - `jwc_properties.ini`
   - `watchdog.ini`

7. Copy the `java.sig` signature file obtained from Qualcomm to the `/sys/mod/java` directory.
8. If you want to enable support for a Java soft reboot on your Qualcomm IoE board, then create the `/sys/mod/reboot_java` directory and drag and drop the following files from the Oracle Java ME Embedded distribution's `reboot_java` folder into that directory.
   - `reboot_java.mif`
   - `reboot_java.mod`
9. Copy the `reboot_java.sig` file obtained from Qualcomm to the `sys/mod/reboot_java` directory.
10. Reset the board by pressing the **RESET KEY** on the board, then wait approximately 40 seconds for the Java VM to start.
11. Start the **Logger** application.
12. Connect to the board using the Connection type: **Brew Devices (COM/DIAG)**, and whichever port matches the Qualcomm HS-USB diagnostics port. Press the **Start Logging** button, and verify that the Java VM is sending logging information to the Logger application by checking for messages that come from the `[JVMStdout]` file name. See Figure 1–3.

Figure 1–3 Brew SDK Logger Application



TimeStamp	Message	FileName	Line Number	Level
2014/04/24 16:37:23 2505	- jdk/dio/DeviceManager.loadFromDriver(), bci=1...	[JVMStdout]	0	3
2014/04/24 16:37:23 2506	- jdk/dio/DeviceManager.open(), bci=313	[JVMStdout]	0	3
2014/04/24 16:37:23 2507	- jdk/dio/DeviceManager.open(), bci=4	[JVMStdout]	0	3
2014/04/24 16:37:23 2508	- com/oracle/mee/systemcontrollersample/Button...	[JVMStdout]	0	3
2014/04/24 16:37:23 2509	- com/oracle/mee/systemcontrollersample/Button...	[JVMStdout]	0	3
2014/04/24 16:37:23 2510	- com/oracle/mee/systemcontrollersample/Syste...	[JVMStdout]	0	3
2014/04/24 16:37:23 2511	- javax/microedition/midlet/MIDletTunnelImpl.call...	[JVMStdout]	0	3
2014/04/24 16:37:23 2512	- com/sun/midp/midlet/MIDletPeer.startApp(), bci...	[JVMStdout]	0	3
2014/04/24 16:37:23 2513	- com/sun/midp/midlet/MIDletStateHandler.startS...	[JVMStdout]	0	3
2014/04/24 16:37:23 2514	- com/sun/midp/main/AbstractMIDletSuiteLoader...	[JVMStdout]	0	3
2014/04/24 16:37:23 2515	- com/sun/midp/main/CldcMIDletSuiteLoader.sta...	[JVMStdout]	0	3
2014/04/24 16:37:23 2516	- com/sun/midp/main/AbstractMIDletSuiteLoader...	[JVMStdout]	0	3
2014/04/24 16:37:23 2517	- com/sun/midp/main/AppIsolateMIDletSuiteLoa...	[JVMStdout]	0	3
2014/04/24 16:37:24 2518	Apr 24 16:37:36 global com.oracle.mee.systemcon...	[JVMStdout]	0	3
2014/04/24 16:37:24 2519	INFO Button listener initialization is failed.	[JVMStdout]	0	3
2014/04/24 16:37:24 2520	Apr 24 16:37:36 global com.oracle.mee.systemcon...	[JVMStdout]	0	3
2014/04/24 16:37:24 2521	INFO Starting state task...	[JVMStdout]	0	3
2014/04/24 16:37:24 2522	Apr 24 16:37:37 global com.oracle.mee.systemcon...	[JVMStdout]	0	3
2014/04/24 16:37:24 2523	WARNING Device interface jdk.dio.gpio.GPIOPin ...	[JVMStdout]	0	3
2014/04/24 16:37:24 2524	Apr 24 16:37:37 global com.oracle.mee.systemcon...	[JVMStdout]	0	3
2014/04/24 16:37:24 2525	INFO State task start is failed.	[JVMStdout]	0	3
2014/04/24 16:37:24 2526	Connection to TaskManager service client closed ...	[JVMStdout]	0	4
2014/04/24 16:37:32 2529	# LOOPBACK IP address: 127.0.0.1	[JVMStdout]	0	3
2014/04/24 16:37:32 2530	# WLAN IP address: 192.168.0.100	[JVMStdout]	0	3
2014/04/24 16:37:32 2531	# Can not obtain PDP 1 IP address. Interface mig...	[JVMStdout]	0	3
2014/04/24 16:37:32 2532	# Can not obtain PDP 2 IP address. Interface mig...	[JVMStdout]	0	3
2014/04/24 16:37:32 2533	# Can not obtain PDP 3 IP address. Interface mig...	[JVMStdout]	0	3

Currently logging to C:\Users\MM.M\Desktop\BrewLogger_140424_163415.log
Log output display active

13. After Java is successfully running on the Qualcomm IoE board, go to the Chapter 2, "Installing and Running Applications on the Qualcomm IoE Board" chapter to learn how to use the tooling features of the Oracle Java ME Embedded software on the board. If Java is not running, then see Chapter 3, "Troubleshooting" to diagnose possible problems.

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 Application Management System (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.

Installing and Configuring the Oracle Java ME SDK 8

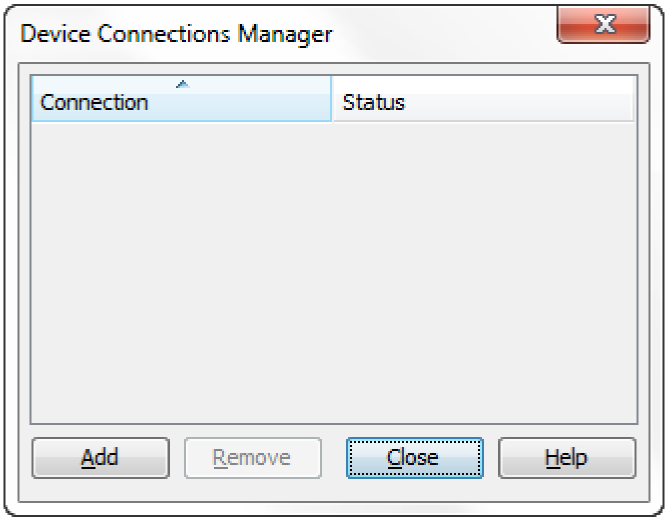
Download and install the Oracle Java ME SDK 8 distribution onto your Windows desktop platform from the Oracle Technology Network website.

`http://www.oracle.com/technetwork/java/javame/javamobile/download/sdk/index.html`

To start the Java ME SDK:

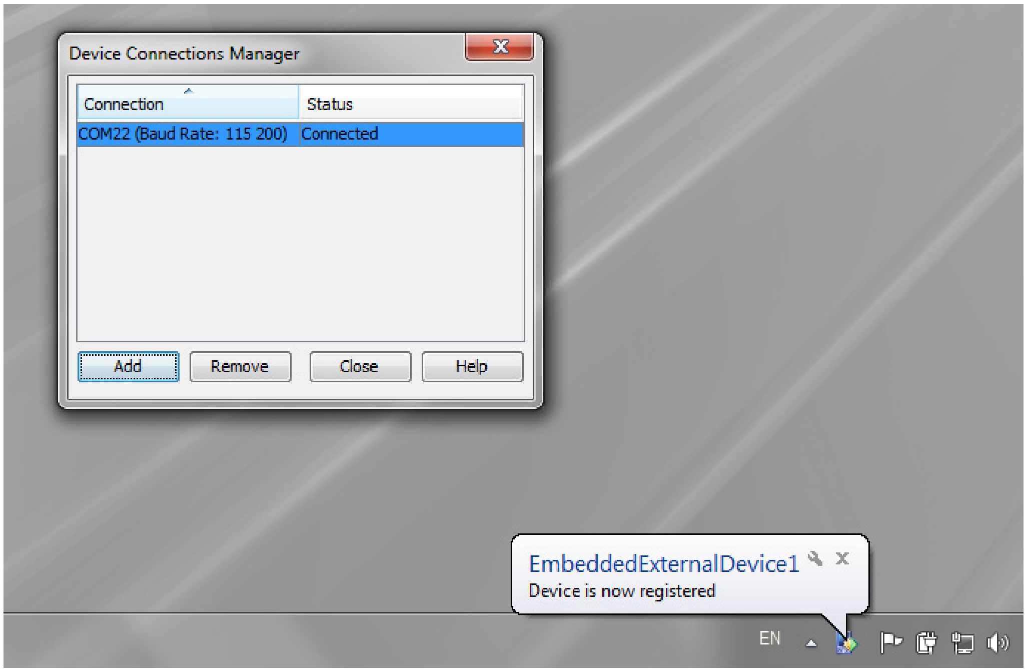
1. Run the **Oracle Java ME SDK Device Manager** (located at *<SDK Installation Folder>*`/bin/device-manager.exe`) by right-clicking its icon in the taskbar and selecting **Manage Device Connections** to display the Device Connections Manager as shown in Figure 1–4.

Figure 1–4 Oracle Java ME SDK Device Connections Manager



2. Restart the Java runtime on the Qualcomm IoE board. At this point, the Oracle Java ME SDK Device Connections Manager should locate the device and report that an EmbeddedExternalDevice1 has now become registered as shown in Figure 1–5.

Figure 1–5 Registering the External Device



Device Connections Manager

Connection	Status
COM22 (Baud Rate: 115 200)	Connected

EmbeddedExternalDevice1

Device is now registered

Note that you can freely reboot the board or restart Java on the board without rebooting the Oracle Java ME SDK Device Connections Manager. However, if you reboot the Device Connections Manager, you must reboot Java or the board as well. If you have problems with the Device Connections Manager connecting to the board, see Chapter 3, "Troubleshooting."

2 Installing and Running Applications on the Qualcomm IoE Board

This chapter describes how to install and run an IMlet on the board, as well as how to debug an IMLet in NetBeans.

> **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*.

Tooling Overview

The Oracle Java ME Embedded platform offers the following tools for managing and monitoring the Qualcomm IoE board:

- A CLI via a terminal emulator program for Application Management System (AMS) commands and for system configuration commands
- A logging interface for obtaining JVM diagnostic information using a console window
- On-Device Tooling (ODT): the ability to install, run, and debug applications from the desktop using the NetBeans IDE

The CLI is integrated in the Developer Agent program that can be found as a JAR file inside the `util` directory of the Oracle Java ME Embedded distribution, named `proxy.jar`. The Developer Agent program can also be used to access the Java Logger output.

There are two options for tooling with the Qualcomm IoE embedded board.

1. Start the Developer Agent program manually as described in Starting the Developer Agent Program Manually.
   - The CLI is available through a PuTTY terminal window on port 65002 (see Using the Command-Line Interface.)
   - The Java Logging is available through a console window used for running the Developer Agent (see Obtaining Java Logs from a Device.)
2. Start the Oracle Java ME Embedded SDK (see Installing and Configuring the Oracle Java ME SDK 8.)
   - The CLI is available through a PuTTY terminal window on port 65002 (see Using the Command-Line Interface.)

- The Java Logging is available via the Output window of the Oracle Java ME Embedded Emulator included with the Oracle Java ME Embedded SDK (see Obtaining Java Logs from a Device.)

Note that tooling works over one physical channel. However, the CLI, logging, and ODT functions all use different ports. The ports for CLI and logging are available to the user, and will be discussed later in this chapter. However, the ports used for ODT are invisible to the user, and used by external development tools.

Starting the Developer Agent Program Manually

Start the Developer Agent program manually only if you want to use the CLI or connect to the Java Logger and do not wish to use the Oracle Java ME Embedded SDK.

1. Open a console window and go to the `util` directory of the Oracle Java ME Embedded distribution.
2. Enter the following command, specifying the COM port that corresponds to the **Qualcomm IoE HT-USB Serial Port**, as reported earlier by the Windows **Device Manager:**

   ```
   java -jar proxy.jar -serial COM22
   ```

If you have trouble with running the Device Manager, please see Chapter 3, "Troubleshooting."

Java Logging Interface Using the Developer Agent Program

To connect to the Java logger, start the Developer Agent program on your desktop computer as described in Starting the Developer Agent Program Manually.

After you are connected, you should see output from the Java Logger.

Connecting to the Java logger displays the logging information from only the Oracle Java ME Embedded platform. However, you can use the Brew MP SDK Logger application, as shown in Chapter 1, to capture logging output from both the Oracle Java ME Embedded system and the Qualcomm IoE board.

Using the Command-Line Interface

The command-line interface is used to issue commands directly to Java runtime.

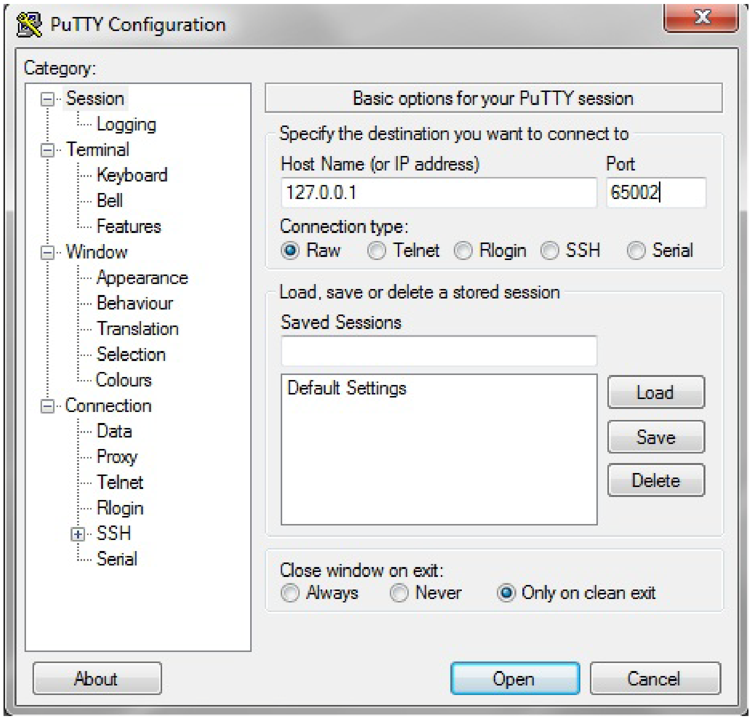
Note that only one instance of the Developer Agent program must be running.

If you are using the Oracle Java ME Embedded SDK, then ensure that the Oracle Java ME SDK 8 Device Connections Manager has already successfully detected the device, as shown earlier in Figure 1–5.

If you do not use the Oracle Java ME Embedded SDK, start the Developer Agent program as described in Starting the Developer Agent Program Manually.

To use the command-line interface, start a PuTTY executable file on your desktop computer. Use this to create a Raw network connection to the address 127.0.0.1 with the port 65002, as shown in Figure 2–1.

Figure 2–1 PuTTY Configuration for CLI Connection



Specify the destination you want to connect to

Host Name (or IP address): 127.0.0.1

Port: 65002

Connection type: ● Raw ○ Telnet ○ Rlogin ○ SSH ○ Serial

Load, save or delete a stored session

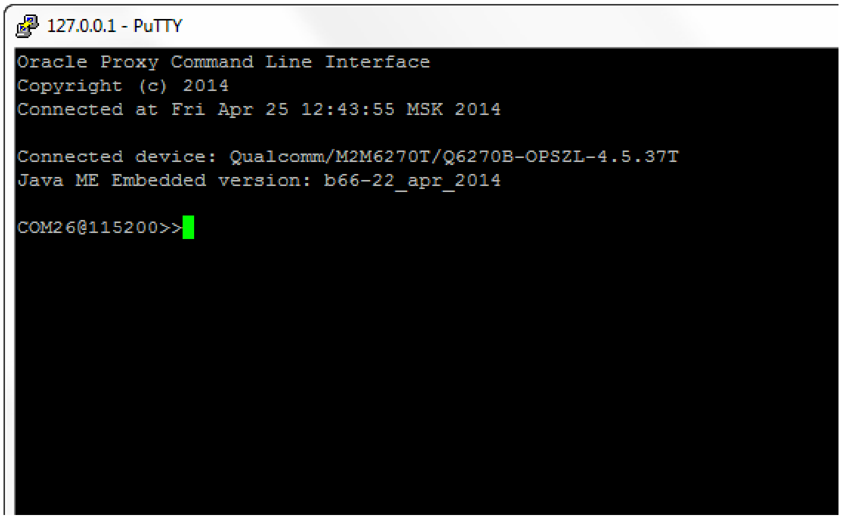
Saved Sessions

Default Settings

Close window on exit: ○ Always ○ Never ● Only on clean exit

The window from the connection provides a CLI, and is shown in Figure 2–2.

Figure 2–2 Oracle Java ME Embedded Command-Line Interface


127.0.0.1 - PuTTY

```
Oracle Proxy Command Line Interface
Copyright (c) 2014
Connected at Fri Apr 25 12:43:55 MSK 2014

Connected device: Qualcomm/M2M6270T/Q6270B-OPSZL-4.5.37T
Java ME Embedded version: b66-22_apr_2014

COM26@115200>>
```

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.

AMS and System Commands

You can use the command-line interface to run numerous AMS and system commands, as shown in Table 2–1, Table 2–2, and Table 2–3.

Table 2–1 AMS CLI Commands

Syntax	Description
`ams-list [INDEX or NAME|VENDOR]`	List all installed IMlet suites and their status or MIDlets in a specific 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> [auth=username[:password]]`	Update the specified suite.
`ams-remove <INDEX or NAME|VENDOR>`	Remove the specified suite.
`ams-run <INDEX or NAME|VENDOR> [MIDLET_ID]`	Run a default suite's MIDlet or the MIDlet specified with `MIDLET_ID` parameter.
`ams-stop <INDEX or NAME|VENDOR> [MIDLET_ID]`	Stop a default suite's MIDlet or the MIDlet specified with the `MIDLET_ID` parameter.
`ams-info <INDEX or NAME|VENDOR>`	Show information about the specific MIDlet.

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

Table 2–2 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 Oracle Java ME Embedded runtime (properties that are set in the `properties.ini` file)
`set-property <NAME> <VALUE>`	Set a property identified by `<NAME>` with the value `<VALUE>`.
`get-property <NAME> [i]`	Return a value of the property identified by `<NAME>`.
`save-properties`	Save properties to an internal storage.
`blacklist -client <NAME>`	Blacklist clients and applications.
`blacklist -app <NAME|VENDOR>`	

Table 2–3 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.
`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.
`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 2–4

Table 2–4 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 2–5.

Table 2–5 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.

Qualcomm IoE board supports the following network related commands, as shown in Table 2–6.

Table 2–6 Qualcomm IoE Specific Commands

Syntax	Description
`net-info`	Show the network information of the system.
`net-set ssid <SSID>`	Set the SSID value for WiFi access.
`net-set passwd <PASSWD>`	Set the password for WiFi access.
`net-set pref <0|1|2|3|4|5>`	Set the network mode preferences. Possible values are: 0: AUTO, 1: NO OP, 2: WLAN Only, 3: GSM/WCDMA only, 4: WCDMA only, 5: GSM/WCDMA/WLAN
`net-set apn <APN>`	Set the APN.
`net-set pdp_authtype <0|1|2>`	Set the APN's auth type: 0: NONE, 1: PAP, 2: CHAP
`net-set pdp_username <USERNAME>`	Reset the PDP user name.
`net-set pdp_password <PASSWORD>`	Reset the PDP password.
`net-reconnect`	Reconnect the network and reboots Java.

Here is a typical example of using the AMS to install, list, run, and remove an Oracle Java ME Embedded application on the board. Note that `/Shared` is a root directory name that can be accessed from Java. Files can be placed in this directory with the help of Qualcomm's Loader tool. However, in the Loader tool, this directory is named `shared`. For more information about mapping the virtual root directory names, see Virtual Root Paths.

```
COM22@115200>> ams-install file:////Shared/hello.jar
<<ams-install,start install,file:///Shared/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

COM22@115200>> 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

COM22@115200>> 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.

```
COM22@115200>> 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.

```
COM22@115200>> ams-remove 0
<<ams-remove,OK,hello removed
```

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

```
COM22@115200>> 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.

```
COM22@115200>> ams-run 1
<<ams-run,OK,started

COM22@115200>> ams-list
<<ams-list,1.netdemo|Oracle,RUNNING
<<ams-list,2.rs232dem|Oracle,RUNNING
<<ams-list,OK,2 suites are installed
```

To check the current WiFi settings of the board, use the net-info command.

```
COM22@115200>> net-info
<<net,info,Address=192.168.1.103
<<net,info,SSID=network001
<<net,info,Preference=0
<<net,info,PDP APN=wap.cingular
<<net,info,PDP Auth Type=0
<<net,info,PDP Auth Username=user
<<net,info,PDP Auth Password=password
<<net,info,OK,success getting info
```

Virtual Root Paths

Virtual root paths are used for access from IMlets. To get access from the Qualcomm tools, these root names are mapped to physical directories as shown in Table 2–7.

Table 2–7 Virtual Root Paths Mapped to Physical Directories

Virtual Root Name	Physical Directory Name
Root/	fs:/sys/mod/java/appdb/root
Shared/	fs:/shared/
MemoryCard/	fs:/mmc1/

Configuring Wi-Fi Networking

WiFi access must first be configured using the CLI. Connect to the CLI using the instructions in the section Using the Command-Line Interface and perform the following steps:

1. Set the SSID of the WiFi network using the command: `net-set ssid` *{SSID Name}*

2. Set the security password of the SSID network using the command `net-set passwd` *{WiFi Password}*. This is necessary only if the security of the WiFi network is enabled.
3. Enter the `net-reconnect` command to apply the settings. Note that the Oracle Java ME SDK 8 Device Connections Manager will temporarily lose its connection to the Qualcomm IoE device while it resets.

After you have completed these steps, you can verify the settings on the board by connecting to the CLI and performing the following command: `net-info`

If the IP address is `0.0.0.0`, then the connection to the WiFi network was not established successfully; check the network settings and try again. If the SSID and password of the network are correct, then try to reset the board to reinitialize the WiFi. You can use "IP Address Periodic Logging" feature to see the IP address that has been assigned.

> **Note:** Each time you make a connection to the Java Logger with the Developer Agent program, you will see logs labeled with the `[NetSetup]` channel. The logs contain information about a connection result to a WiFi access point. If you cannot connect to the Java Logger, try connecting to the board with Brew MP Loader to find the same logs in the `/shared/netlog.txt` file.

If you see a valid IP address, then the network is configured successfully.

Finally, restart Java again using the `net-reconnect` command and connect to port 65000 and 65002 using the IP address that has been assigned to the board.

IP Address Periodic Logging

The Oracle Java ME Embedded implementation has the ability to log the IP address that has been assigned to the board. The log can be seen through a console window and Brew MP Logger application.

The behavior of this feature is controlled by the `com.oracle.periodic.logging.interval` property that accepts the values counter in milliseconds. By default, the logging period is set to 10 seconds (10000 milliseconds.) To disable the periodic logging, set the value of the property to 0.

Setting Up the System Time

To run the signed IMlets on the IoE board, ensure that the system time is set on the board.

By default, the system time on the IoE board is set automatically by the operating system. Because the current time is obtained from a cellular network, a SIM card must be plugged in to the board's SIM-card slot.

However, an automatic setup does not work in the following cases:

- A SIM card is not available.
- A network carrier does not provide the current time data.

If there are any problems with the automatic setup, then set the system time manually either by using the CLI or editing the `jwc_properties.ini` file.

Manual Setup of the System Time Using CLI

1. Open the CLI.
2. Run the following commands:

```
set-property system.time.value yyyy/mm/dd hh:mm:ss GMT
set-property system.time.update true
```

3. Execute either the `save-properties` with a subsequent board restart or `shutdown -r` command. In the latter case, the property values will be saved automatically.

Manual Setup of the System Time in the `jwc_properties.ini` file

1. Edit the following properties:

```
system.time.value = yyyy/mm/dd hh:mm:ss GMT
system.time.update = true
```

2. Restart Java on the Qualcomm IoE board.

> **Note:** The dates before the GPS epoch, such as 06.01.1980 00:00:00, are not supported due to the Brew MP API restrictions.

Reconnecting to Access Points

You can configure a network type of an access point to connect to and other specific access point preferences by using the `system.network.*`, `system.wlan.*`, and `system.netsetup.*` properties.

An ability to reconnect to a WiFi access point is controlled by the following WiFi rescanning process properties:

- `system.netsetup.wifi.rescan.enabled`
  - if `true`, then Java starts rescanning the WLAN to find an access point with the SSID specified by the `system.wlan.ap.ssid` property in the following cases:
    - upon the board's startup, if the access point with specified SSID is not available and `system.network.pref` property is set to `2`
    - when an access point with the specified SSID becomes unavailable and `system.network.pref` property is set to 5, 2, or 0 (`AUTO`)
  - if `false`, the Java makes no attempts to reconnect to an access point. However, the Brew MP operating system might try to reconnect to the previously connected access point at some unknown time.
- `system.netsetup.wifi.rescan.timeout`: a frequency in millis with which the WiFi rescanning occurs if the access point with the specified SSID is not available
- `system.netsetup.wifi.rescan.attempts:` a number of attempts during which Java tries to reconnect to the access point with the specified SSID

After the connection to a 3GPP network was lost and then restored, there is no guarantee that the 3GPP network becomes available to the application. An ability to reconnect to a 3GPP system access point is a network carrier-specific.

Note that Java starts with a delay that depends on a network initialization.

Using NetBeans with the Qualcomm IoE Board

Installing and running IMlet projects on the Qualcomm IoE board using the NetBeans IDE requires the following software:

- NetBeans IDE 8.0 with Oracle Java ME, which can be downloaded from `https://netbeans.org/`.
- Oracle Java ME SDK 8
- Oracle Java ME SDK 8 NetBeans Plug-in

Installing the Oracle Java ME SDK 8 Plug-in for NetBeans

After installing NetBeans, follow these steps to install the remaining software:

1. Ensure that Oracle Java ME is enabled in NetBeans. This can be done by clicking **Tools** then **Plugins** and clicking the **Installed** pane. Activate the Java ME plugin if it is not already activated.
2. Install the Oracle Java ME SDK 8 distribution, if you have not done so already. See the *Oracle Java ME SDK Developer's Guide* for details.
3. Install the Oracle Java ME SDK 8 NetBeans plug-in. This is a downloadable ZIP file that consists of a number of NetBeans modules (`.nbm` files) that can be added by clicking **Tools** then **Plugins.** Select the **Downloaded** pane. Unzip the plugin file, and add all of the `.nbm` files to NetBeans. The Oracle Java ME SDK 8 NetBeans plug-ins are required to interface with the Device Selector and connect to the board.
4. Ensure that the Oracle Java ME SDK 8 appears in the list of Java ME platforms. In the NetBeans IDE, click **Tools** then **Java Platforms**. If the Oracle Java Platform Micro Edition SDK 8 does not appear in the list of Java ME platforms, then follow these steps:
   1. Click **Add Platform**.
   2. Select **Java ME CLDC Platform Emulator** and click **Next**.
   3. Select the folder where the Oracle Java ME SDK 8 distribution resides and follow the instructions to install it. Then, click **Finish** to close the dialog.
5. Ensure that the Qualcomm IoE board has the Oracle Java ME Embedded distribution. See Chapter 1, "Installing Oracle Java ME Embedded Software on the Qualcomm IoE Board" for more information about how to install the runtime distribution on the Qualcomm IoE board.

Assigning the Qualcomm IoE 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, then follow these steps:

1. Right-click your project and select **Properties**.
2. Select the **Platform** category on the properties window.
3. Select the entry that represents the board (**EmbeddedExternalDevice1**) from the device list.

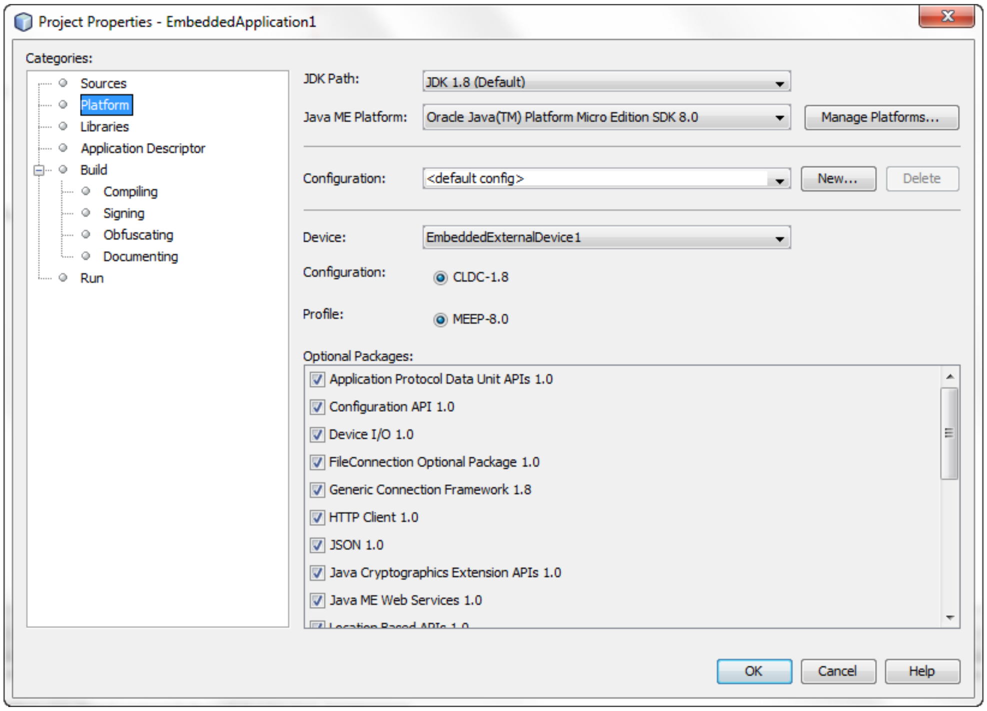
If you are creating a new NetBeans project, then follow these steps:

1. Click **File** then **New Project**.

2. Select the **Java ME Embedded** category and **Java ME Embedded Application** in the Projects list. Click **Next**.
3. Provide a project name and click **Next**. Ensure that the **Create MIDlet** option is selected.
4. Ensure that the Java ME platform is **Oracle Java Micro Edition SDK 8.0**. Then, select the entry that represents the board (**EmbeddedExternalDevice1**) from the device list and click **Finish**.

The configured Platform dialog is shown in Figure 2–3. After you assign the board to your project, the IMlets run on the board instead of on the emulator when you click **Run Project** on the NetBeans IDE.

Figure 2–3 NetBeans Platform Properties Dialog



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

Sample Source Code

After the project is created, use the following source code for the default `IMlet.java` source file.

```
package embeddedapplication1;

import jdk.dio.DeviceManager;
import jdk.dio.gpio.GPIOPin;
import java.io.IOException;
import javax.microedition.midlet.*;


public class EmbeddedApplication1 extends MIDlet {

    public void startApp() {

        try {
            GPIOPin pin = (GPIOPin)DeviceManager.open(14);
            for (int i = 0; i < 10; i++) {
                pin.setValue(true);
                Thread.sleep(1000);
                pin.setValue(false);
                Thread.sleep(1000);
            }

            pin.close();

        } catch (IOException ex) {
            ex.printStackTrace();
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }

    }

    public void pauseApp() {
    }

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

This sample code gets an object that represent GPIO pin 14 from the `DeviceManager` instance, and sets it from low to high at intervals of one second. This has the effect of blinking one of the LEDs on the Qualcomm IoE board. 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`

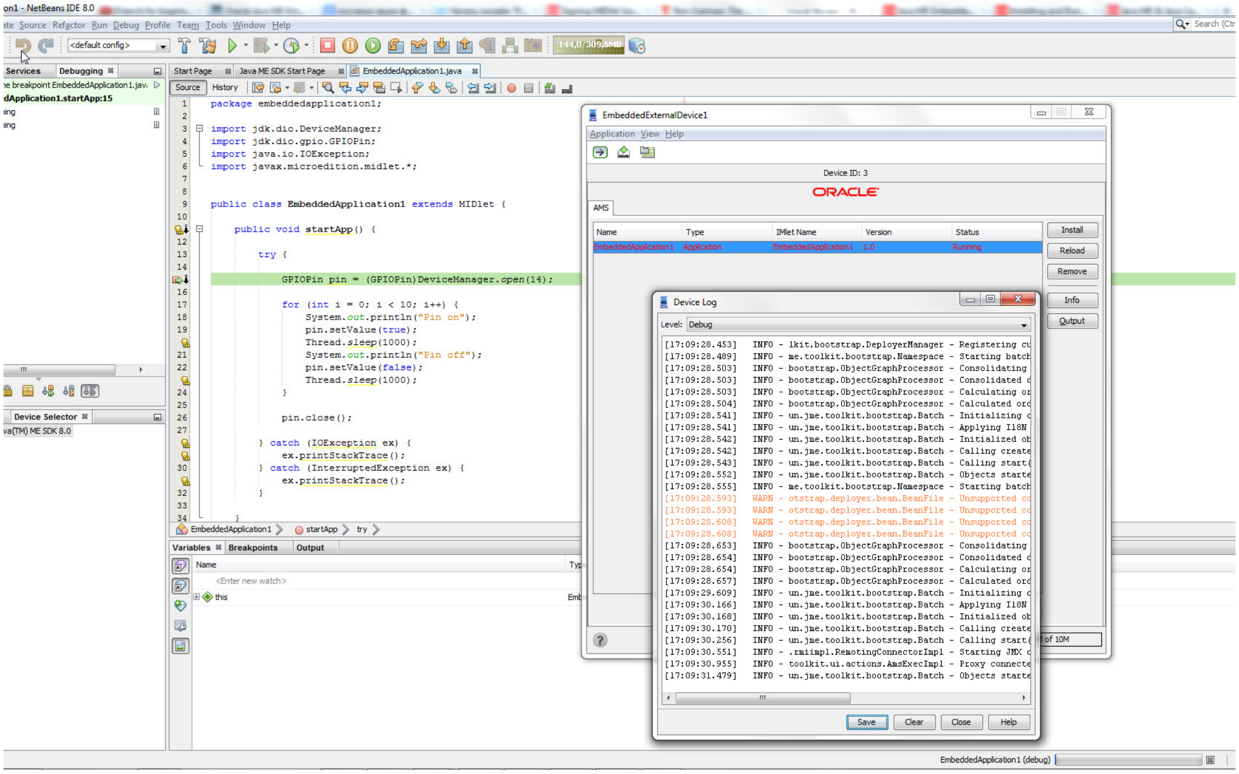
Debugging an IMlet on the Qualcomm IoE Board

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. Click **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 running at your breakpoint, as shown in Figure 2–4.

Figure 2–4 Debugging an IMlet on the Qualcomm IoE Board Using NetBeans



```
package embeddedapplication1;

import jdk.dio.DeviceManager;
import jdk.dio.gpio.GPIOPin;
import java.io.IOException;
import javax.microedition.midlet.*;


public class EmbeddedApplication1 extends MIDlet {

    public void startApp() {

        try {

            GPIOPin pin = (GPIOPin)DeviceManager.open(14);

            for (int i = 0; i < 10; i++) {
                System.out.println("Pin on");
                pin.setValue(true);
                Thread.sleep(1000);
                System.out.println("Pin off");
                pin.setValue(false);
                Thread.sleep(1000);
            }

            pin.close();

        } catch (IOException ex) {
            ex.printStackTrace();
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }

    }
```

Name	Type	IMlet Name	Version	Status
EmbeddedApplication1	Application	EmbeddedApplication1	1.0	Running

```
[17:09:28.453]  INFO - lkit.bootstrap.DeployerManager - Registering cu
[17:09:28.489]  INFO - me.toolkit.bootstrap.Namespace - Starting batch
[17:09:28.503]  INFO - bootstrap.ObjectGraphProcessor - Consolidating
[17:09:28.503]  INFO - bootstrap.ObjectGraphProcessor - Consolidated d
[17:09:28.503]  INFO - bootstrap.ObjectGraphProcessor - Calculating or
[17:09:28.504]  INFO - bootstrap.ObjectGraphProcessor - Calculated ord
[17:09:28.541]  INFO - un.jme.toolkit.bootstrap.Batch - Initializing c
[17:09:28.541]  INFO - un.jme.toolkit.bootstrap.Batch - Applying I18N
[17:09:28.542]  INFO - un.jme.toolkit.bootstrap.Batch - Initialized ob
[17:09:28.542]  INFO - un.jme.toolkit.bootstrap.Batch - Calling create
[17:09:28.543]  INFO - un.jme.toolkit.bootstrap.Batch - Calling start(
[17:09:28.552]  INFO - un.jme.toolkit.bootstrap.Batch - Objects starte
[17:09:28.555]  INFO - me.toolkit.bootstrap.Namespace - Starting batch
[17:09:28.593]  WARN - otstrap.deployer.bean.BeanFile - Unsupported co
[17:09:28.593]  WARN - otstrap.deployer.bean.BeanFile - Unsupported co
[17:09:28.608]  WARN - otstrap.deployer.bean.BeanFile - Unsupported co
[17:09:28.608]  WARN - otstrap.deployer.bean.BeanFile - Unsupported co
[17:09:28.653]  INFO - bootstrap.ObjectGraphProcessor - Consolidating
[17:09:28.654]  INFO - bootstrap.ObjectGraphProcessor - Consolidated d
[17:09:28.654]  INFO - bootstrap.ObjectGraphProcessor - Calculating or
[17:09:28.657]  INFO - bootstrap.ObjectGraphProcessor - Calculated ord
[17:09:29.609]  INFO - un.jme.toolkit.bootstrap.Batch - Initializing c
[17:09:30.166]  INFO - un.jme.toolkit.bootstrap.Batch - Applying I18N
[17:09:30.168]  INFO - un.jme.toolkit.bootstrap.Batch - Initialized ob
[17:09:30.170]  INFO - un.jme.toolkit.bootstrap.Batch - Calling create
[17:09:30.256]  INFO - un.jme.toolkit.bootstrap.Batch - Calling start(
[17:09:30.551]  INFO - .rmiimpl.RemotingConnectorImpl - Starting JMX c
[17:09:30.955]  INFO - toolkit.ui.actions.AmsExecImpl - Proxy connecte
[17:09:31.479]  INFO - un.jme.toolkit.bootstrap.Batch - Objects starte
```

Figure 2–4 shows an entire NetBeans debugging environment that allows the programmer to run 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 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`

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, please see the *Device I/O API 1.0* specification at:

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

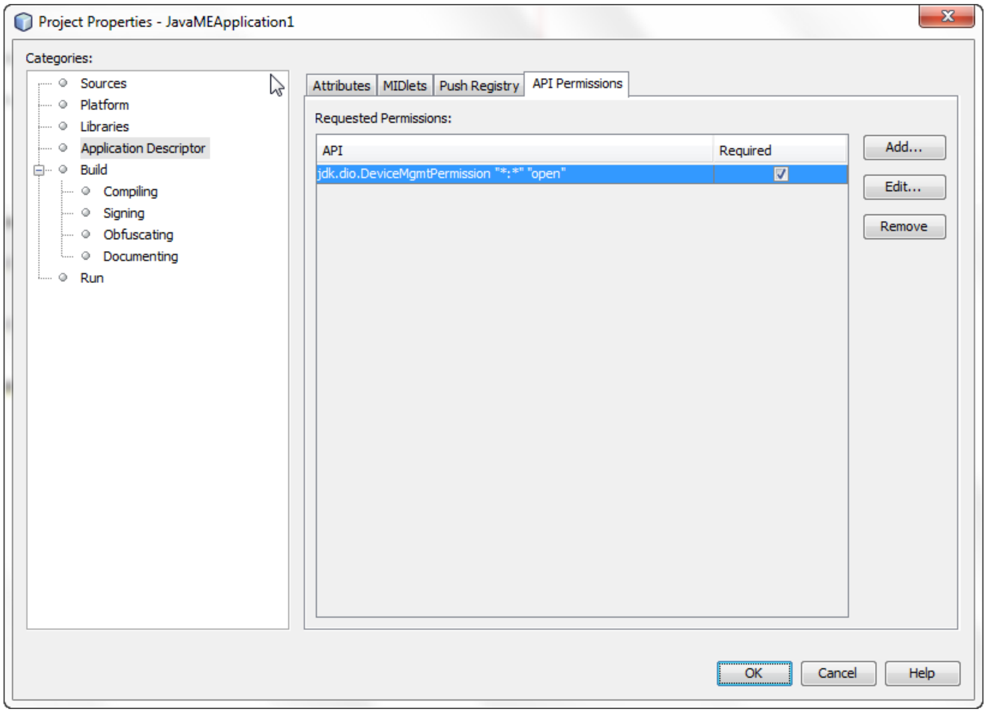
Signing the Application with API Permissions

First, the JAD file must have the proper API permissions. Follow these steps how 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 2–5.

4. Click **OK** to close the project properties dialog.

Figure 2–5 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 permission:

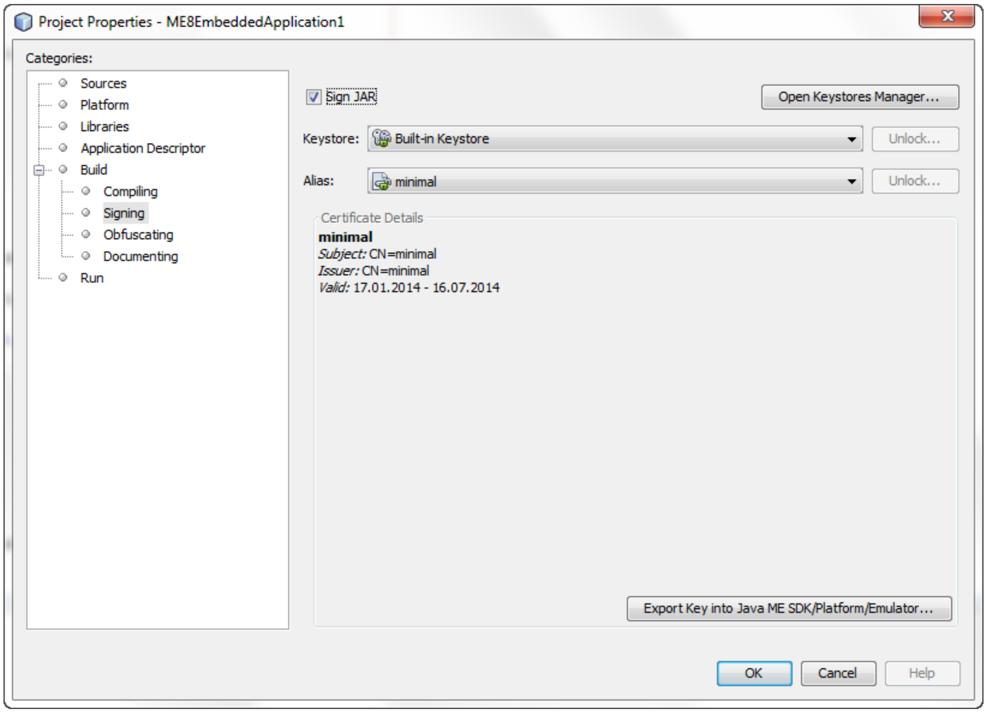
```
MIDlet-Permission-1: jdk.dio.DeviceMgmtPermission "*:*" "open"
```

Method #1: Signing Application Using the NetBeans IDE

This NetBeans IDE enables developers both to sign the applications with a local certificate and upload the certificate on the device. Use the following procedure.

1. Right-click the project name and select **Properties**.
2. Under the **Build** category, click **Signing**.
3. Select the **Sign JAR** check box and specify a certificate to sign with as shown in Figure 2–6.

Figure 2–6 Signing Application JAR with NetBeans



☑ Sign JAR

Keystore: Built-in Keystore

Alias: minimal

Certificate Details

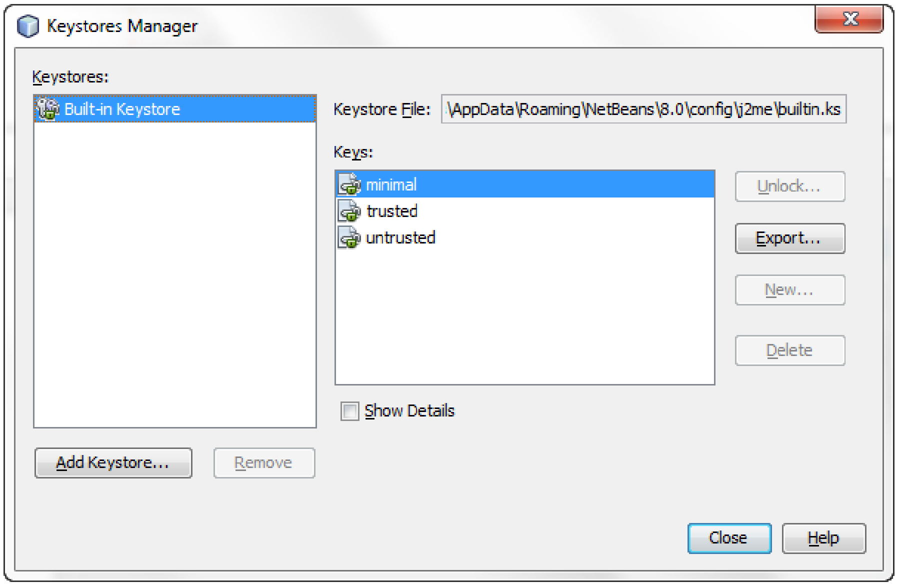
minimal
Subject: CN=minimal
Issuer: CN=minimal
Valid: 17.01.2014 - 16.07.2014

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

---

4. Click the **Open Keystores Manager** button.
5. Select the key and click **Export** as shown in Figure 2–7.

Figure 2–7 Keystores Manager Window

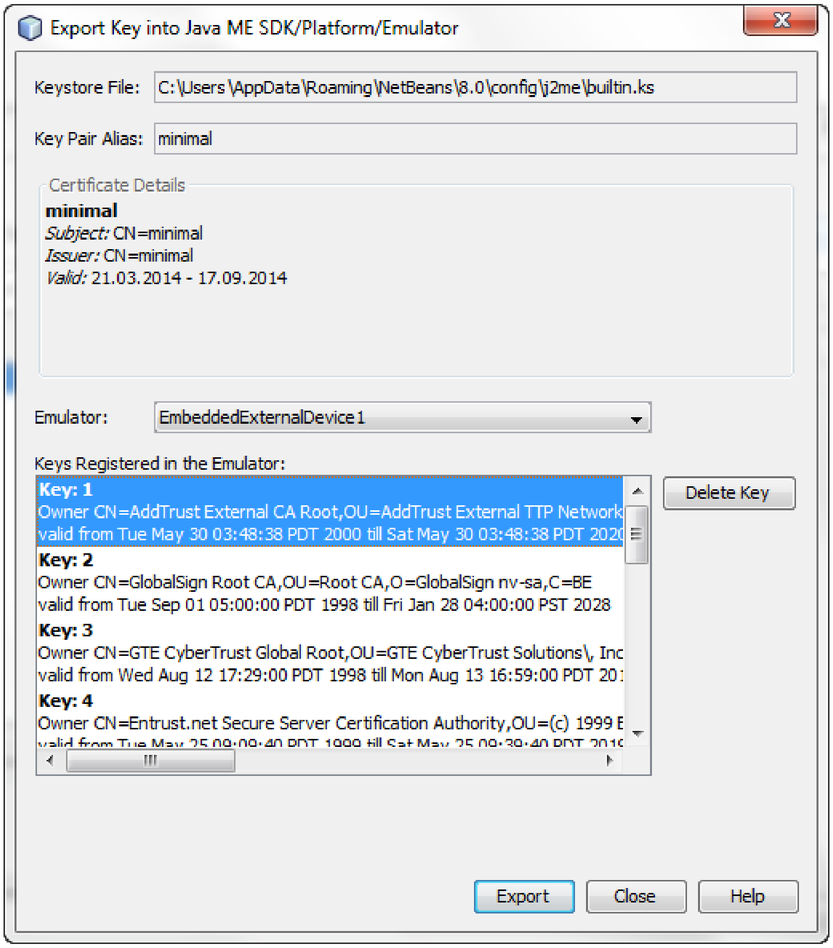


Keystore File: \AppData\Roaming\NetBeans\8.0\config\j2me\builtin.ks

□ Show Details

6. In the Export Key window, select the EmbeddedExternalDevice1, select the certificate, and click **Export** as shown in Figure 2–8.

Figure 2–8 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 202
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 20
Key: 4
Owner CN=Entrust.net Secure Server Certification Authority,OU=(c) 1999 E

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

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

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

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

Method #2: Signing Application with a Local Certificate

This method is the preferred route for applications that are widely distributed. Follow these steps 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 Java SE JDK:

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

Note: You must specify the full host name in the `CN=thehost` parameter.

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 `appdb/_main.ks` keystore file from the Qualcomm IoE over to the desktop using the Loader tool. Run the following command with 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 Qualcomm IoE board using the Loader tool.

Use the following commands to sign your application before deploying it to the Qualcomm IoE 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. This method should be used only for development and debugging. Final testing must be done using a real certificate as described in method #1.

1. To use `NullAuthenticationProvider`, set the following property in the `jwc_properties.ini` file on the Qualcomm IoE board:

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

2. Restart the Java runtime.

Installing and Running an IMlet Using the AMS CLI

If you are not using an IDE, then you can still use the Oracle Java ME Embedded 8 CLI to install an application. Connect to the device at port 65002, and install and run the IMlet manually. For example:

```
COM22@115200>> ams-install file:///Shared/hello.jar
<<ams-install,start install,file:///Shared/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

COM22@115200>> ams-list
<<ams-list,0.hello|Oracle,STOPPED
<<ams-list,OK,1 suites are installed

COM22@115200>> ams-run 0
<<ams-run,OK,started

COM22@115200>> ams-list
<<ams-list,o.hello|Oracle,RUNNING
<<ams-list,OK,1 suites are installed
```

See "Using the Command-Line Interface" for more details.

Obtaining Java Logs from a Device

There are multiple ways in Oracle Java ME 8 to obtain a device log. Three ways to view the device log are:

1. Using an SDK Output Console window.

   - Using the NetBeans IDE:

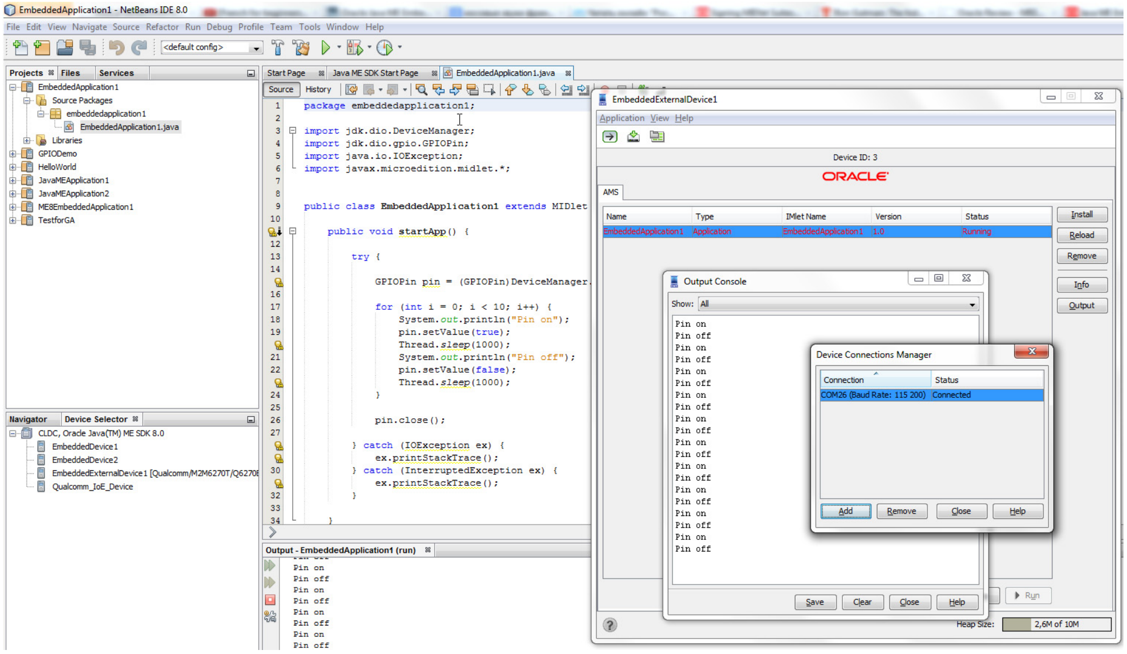
     – Run the NetBeans IDE.

     – Select an IMlet in the Projects window and run it.

       The NetBeans IDE opens the EmbeddedExternalDevice1 window.

     – Click the **Output** button. The log is available in the SDK EmbeddedExternalDevice1 Output Console window shown in Figure 2–9.

Figure 2–9 Java Logging Through the SDK Output Console Using NetBeans IDE



```
package embeddedapplication1;

import jdk.dio.DeviceManager;
import jdk.dio.gpio.GPIOPin;
import java.io.IOException;
import javax.microedition.midlet.*;


public class EmbeddedApplication1 extends MIDlet

    public void startApp() {

        try {

            GPIOPin pin = (GPIOPin)DeviceManager.

            for (int i = 0; i < 10; i++) {
                System.out.println("Pin on");
                pin.setValue(true);
                Thread.sleep(1000);
                System.out.println("Pin off");
                pin.setValue(false);
                Thread.sleep(1000);
            }

            pin.close();

        } catch (IOException ex) {
            ex.printStackTrace();
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }

    }
```

Output - EmbeddedApplication1 (run)

```
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
```

Name	Type	IMlet Name	Version	Status
EmbeddedApplication1	Application	EmbeddedApplication1	1.0	Running

```
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
```

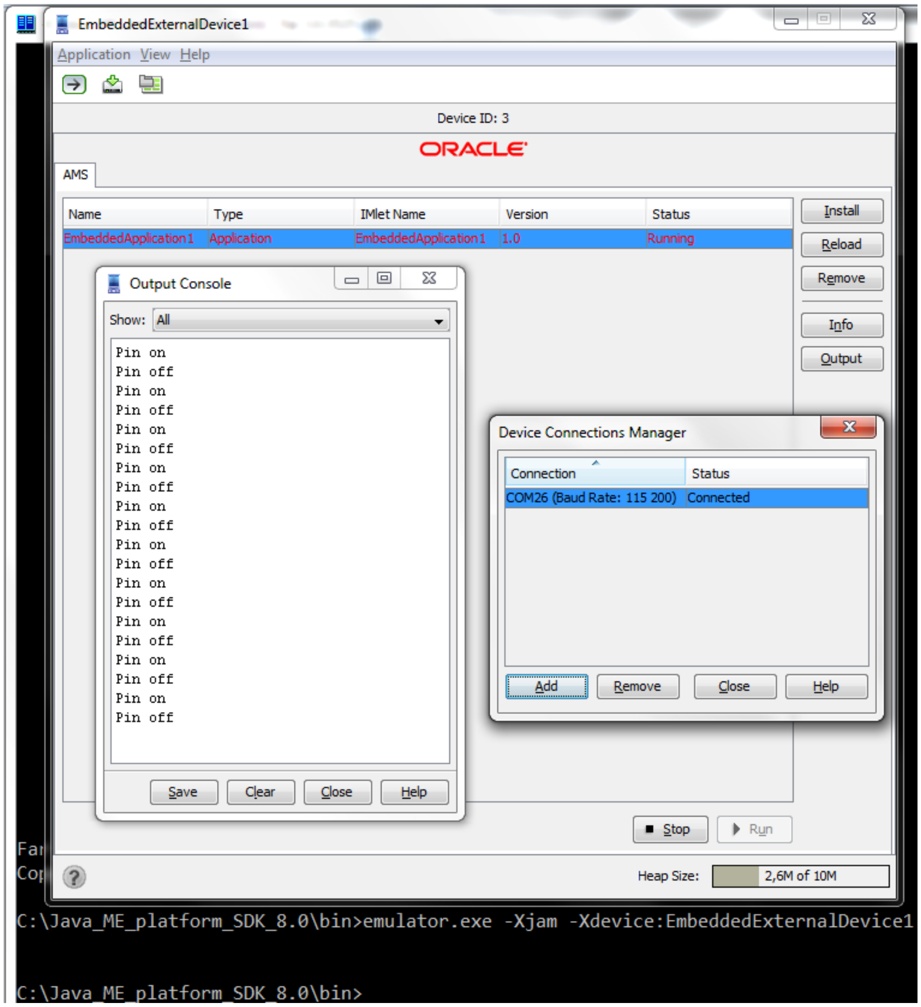
Connection	Status
COM26 (Baud Rate: 115 200)	Connected

- Without the NetBeans IDE:
  - Run the SDK Device Connections Manager located at <*SDK Installation Folder*>`/bin/device-manager.exe`.
  - Wait until the device connection status displays **Connected**.
  - Run the SDK EmbeddedExternalDevice1 by using the following command:

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

  - Install and run an IMlet using the GUI of the SDK EmbeddedExternalDevice1 window.
  - Click the **Output** button. The log is available in the SDK EmbeddedExternalDevice1 Output Console window shown in Figure 2–10.

Figure 2–10 Java Logging Using the SDK Output Console



Name	Type	IMlet Name	Version	Status
EmbeddedApplication1	Application	EmbeddedApplication1	1.0	Running

```
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
```

Connection	Status
COM26 (Baud Rate: 115 200)	Connected

```
C:\Java_ME_platform_SDK_8.0\bin>emulator.exe -Xjam -Xdevice:EmbeddedExternalDevice1

C:\Java_ME_platform_SDK_8.0\bin>
```

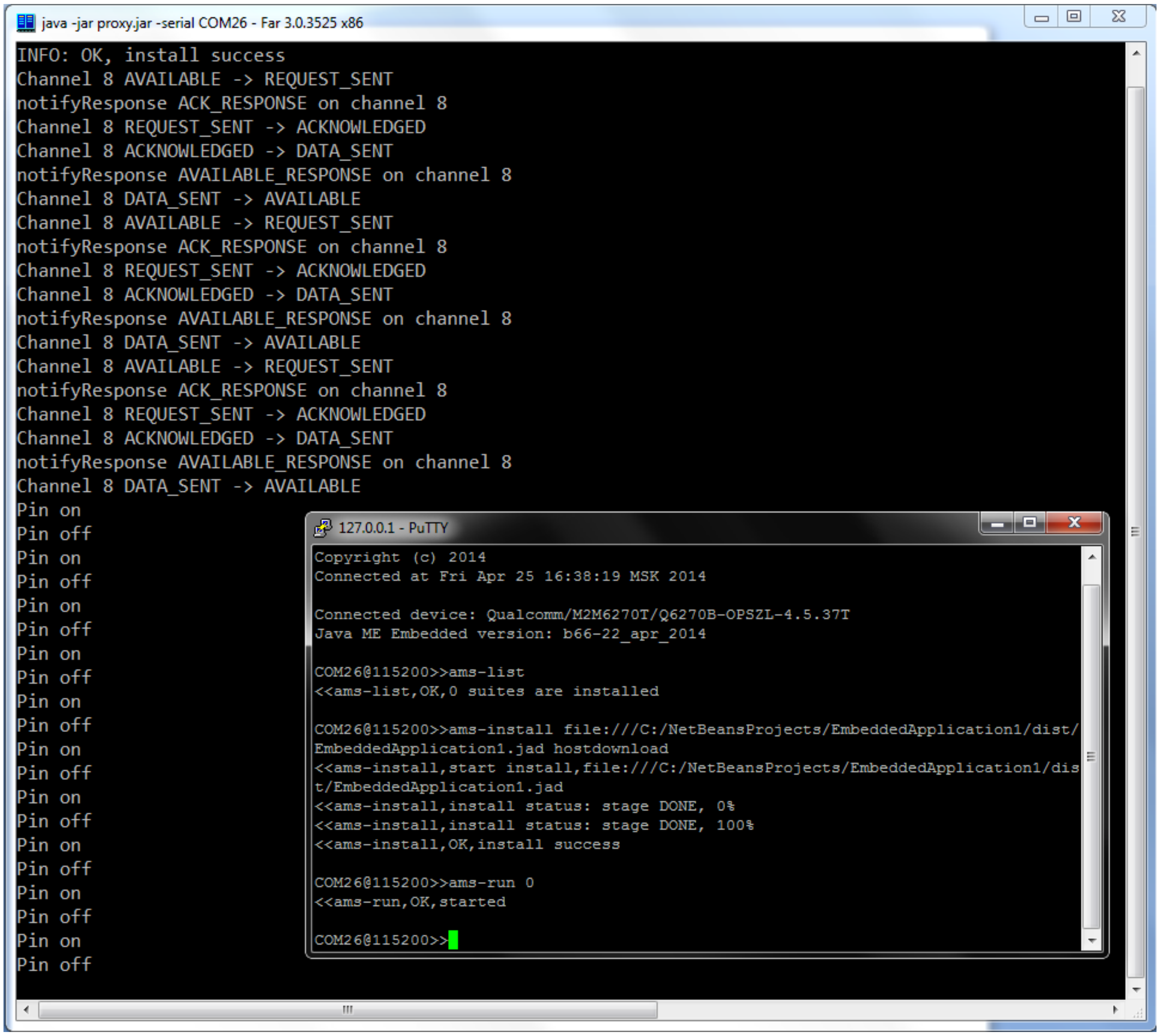
2. Using a console application such as Windows Command Line or Far. In this case, you must run the Developer Agent program manually (see Starting the Developer Agent Program Manually.)

- Start a console application and enter the following command specifying the COM port that corresponds to the Qualcomm IoE HS-USB serial port:

  ```
  java -jar proxy.jar -serial COM22
  ```

- Install and run IMlets using the CLI.
- The log will be available in the same console window shown in Figure 2–11.

Figure 2–11 Java Logging Using a Console Application



```
INFO: OK, install success
Channel 8 AVAILABLE -> REQUEST_SENT
notifyResponse ACK_RESPONSE on channel 8
Channel 8 REQUEST_SENT -> ACKNOWLEDGED
Channel 8 ACKNOWLEDGED -> DATA_SENT
notifyResponse AVAILABLE_RESPONSE on channel 8
Channel 8 DATA_SENT -> AVAILABLE
Channel 8 AVAILABLE -> REQUEST_SENT
notifyResponse ACK_RESPONSE on channel 8
Channel 8 REQUEST_SENT -> ACKNOWLEDGED
Channel 8 ACKNOWLEDGED -> DATA_SENT
notifyResponse AVAILABLE_RESPONSE on channel 8
Channel 8 DATA_SENT -> AVAILABLE
Channel 8 AVAILABLE -> REQUEST_SENT
notifyResponse ACK_RESPONSE on channel 8
Channel 8 REQUEST_SENT -> ACKNOWLEDGED
Channel 8 ACKNOWLEDGED -> DATA_SENT
notifyResponse AVAILABLE_RESPONSE on channel 8
Channel 8 DATA_SENT -> AVAILABLE
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
Pin on
Pin off
```

```
Copyright (c) 2014
Connected at Fri Apr 25 16:38:19 MSK 2014

Connected device: Qualcomm/M2M6270T/Q6270B-OPSZL-4.5.37T
Java ME Embedded version: b66-22_apr_2014

COM26@115200>>ams-list
<<ams-list,OK,0 suites are installed

COM26@115200>>ams-install file:///C:/NetBeansProjects/EmbeddedApplication1/dist/
EmbeddedApplication1.jad hostdownload
<<ams-install,start install,file:///C:/NetBeansProjects/EmbeddedApplication1/dis
t/EmbeddedApplication1.jad
<<ams-install,install status: stage DONE, 0%
<<ams-install,install status: stage DONE, 100%
<<ams-install,OK,install success

COM26@115200>>ams-run 0
<<ams-run,OK,started

COM26@115200>>
```

3. Using the Brew MP Logger application.
   - Start the Brew MP Logger application.
   - When the Logger application starts, connect to the board using the Connection type: **Brew Devices (COM/DIAG)**, and whichever port matches the Qualcomm HS-USB diagnostics port.
   - Connect to the board, click the **Start Logging** button, and verify that the Java VM is sending logging information to the Logger application by checking for messages that come from the [JVMStdout] file name.

Note: Options 1 and 2 are mutually exclusive because only one instance of the Developer Agent program can be run. Option 3 can be used both independently and in parallel with either option 1 or 2.

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 Qualcomm IoE 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 two categories:

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

Starting Oracle Java ME Embedded Software on the Board

Table 3–1 contains information about problems and solutions when starting the runtime on the board.

Table 3–1 Problems and Solutions: Starting Oracle Java ME Embedded Software on the Board

Problem	Cause	Solution
Windows does not recognize the board when connected using USB.	The USB drivers are not loaded.	See Chapter 1, "Installing Oracle Java ME Embedded Software on the Qualcomm IoE Board" for more information about installing the USB drivers for the Qualcomm IoE board.
Windows does not recognize the board when connected using USB.	The board is not powered on.	Press the PWR KEY button on the board.
Oracle Java ME Embedded fails to initialize the network on the board.	The network configuration is incorrect.	Verify that the network connection on the board is correct. Ensure that the board is using DHCP to obtain an IP address.
(*continued)*	The network configuration on the WiFi access point is incorrect or unsupported.	Verify that WiFi SSID broadcasting is enabled on the router.
(continued)	The WPA2-PSK authentication algorithm is not working correctly on some routers.	Try to manually set the authentication algorithm to WPA-PSK or disable the security checking.

Table 3–1 (Cont.) Problems and Solutions: Starting Oracle Java ME Embedded Software on the Board

Problem	Cause	Solution
There are no Java logs in the Brew MP Logger application. However, the device has been recognized successfully by the Logger and Loader applications.	The Oracle Java ME Embedded platform did not start. The `java.sig` file was not updated or is invalid for Oracle Java ME Embedded 8.	Examine the netsetup logs. See Configuring Wi-Fi Networking in Chapter 2 for additional information. If there is no `netlog.txt` file or it is incorrect, then ensure that the Oracle Java ME Embedded application has been updated and the `java.sig` file is valid. The signature file from previous versions of the Oracle Java ME Embedded platform will not work correctly. If the `netlog.txt` file is present and is correct, then ensure that the Oracle Java ME Embedded application in the `/sys/mod/java` directory has been updated and the `java.sig` file is valid. The Java signature file from the previous version must also be updated for version 8.
The board is not detected by the Device Connections Manager when connecting to the board in serial mode.	Varies. See **Solution** column.	If you have some issues with connecting to the Device Connections Manager and the board, then examine the file <*USER_HOME_ DIR*>`/javame-sdk/8.0/log/sos-proxy.log`. **1.** If the last line in the output looks similar to "`Open COM`*{Number}*", then ensure that you have specified the correct COM port. If the port is incorrect, then select the proper one by specifying it in Device Connections Manager and restart the board. Note that is sometimes take a minute or more to boot Java after the board is powered on. This is because of WiFi related settings that are performed during the launch time. If you do not use WiFi (3G), then you can disable the network setup with net-related commands. In addition, there is a "`system.netsetup.timeout`" property that configures the timeout to start Java after the network initialization has been started. If the COM port is correct and more than a minute has passed after the board is powered on, then try to reboot both Device Connections Manager and the board **2.** If the last line in the output is *not* "`Open COM`*{Number}*", verify that you specified the correct port numbers to connect to the CLI or the logger, then try to reboot both the Device Connections Manager and the board
The Developer Agent does not start manually and throws a BindException: "Address already in use: JVM_Bind"	The NetBeans IDE or the Device Connection Manager is running on your desktop host computer.	**1.** Close the NetBeans IDE. **2.** Close the Device Connections Manager. **3.** Start the Developer Agent program again.

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

Table 3–2 contains information about problems and solutions when using the board with the Oracle Java ME SDK and the NetBeans IDE:

Table 3–2 Problems and Solutions: Oracle Java ME SDK and the NetBeans IDE

Problem	Cause	Solution
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.
The current time and date are invalid.	The time and date on the Qualcomm IoE board are configured automatically only if a valid SIM card is inserted and a carrier provides current time data.	Insert a valid SIM card. If the date and time are still wrong, see Chapter 2, "Setting Up the System Time" to set the system time manually.
A signed IMlet will not install. The AMS gives a return code that the certificate or the authentication is invalid.	The certificate is invalid or it is not added to the keystore.	Check the certificate. Refer to Signing the Application with API Permissions in Chapter 2, "Installing and Running Applications on the Qualcomm IoE Board" for details.
(*continued)*	The date and time on the board were configured incorrectly.	Insert a valid SIM card. If the date and time are still wrong, see Chapter 2, "Setting Up the System Time", to set the system time manually.

A

Device I/O Preconfigured List

This appendix provides information about the various peripheral ports and buses for the Qualcomm IoE embedded board, as well as device mappings and important notes, 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.

> **Note:** Power Management, and MMIO are not supported on the Qualcomm IoE embedded board.

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

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

The names and IDs for specific devices can be found in the tables below 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 peripheral with a `DeviceManager`.
- **Device Name** - the string name of a peripheral that can be used to open it by name with `DeviceManager`.
- **Mapped** - all hardware related information regarding a peripheral, such as physical location, mapping, or port. This information enables the user to find out 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 in order to open the peripheral by ID or name. The configuration can be used to open the peripheral using the `DeviceManager` with the appropriate configuration.

AT Devices

The following AT devices are pre-configured.

Device ID	Device Name	Mapped	Configuration

800	DEFAULT	Brew's AT command interface	`controllerNumber = DeviceConfig.DEFAULT`
			`channelNumber = DeviceConfig.DEFAULT`

For a complete list of AT commands that can be used, see the *Qualcomm IoE Development Platform User Guide*.

Please note the following when using AT commands:

- Some AT commands require a SIM card to test. (for example, "`AT+CPBW`" or "`AT+CMUX`")
- With the `AT+CPBW` command, the valid form is "`AT+CPBW=?`" or "`AT+CPBW=<num>`". "`AT+CPBW?`" is an invalid form.
- `UnsolicitedResponseHandler` is not supported.

Analog-to-Digital Converter (ADC) Devices

The following Analog-to-Digital (ADC) devices are pre-configured.

Device ID	Device Name	Mapped	Configuration

100	VTHERM_N	ADT7481	`controllerNumber = DeviceConfig.DEFAULT`
			`channelNumber = 0`
			`resolution = DeviceConfig.DEFAULT`
			`samplingInterval = DeviceConfig.DEFAULT`
			`samplingTime = DeviceConfig.DEFAULT`

101	HKAIN1	Any pin from J10 header.
		Pin number is chosen according to
		the configuration of ADC multiplexer
			`controllerNumber = DeviceConfig.DEFAULT`
			`channelNumber = 1`
			`resolution = DeviceConfig.DEFAULT`
			`samplingInterval = DeviceConfig.DEFAULT`
			`samplingTime = DeviceConfig.DEFAULT`

Note the following:

- The channel number set as `DeviceConfig.DEFAULT` is interpreted as 1, that is, the ADC channel connected to a multiplexer on the Qualcomm IoE board will be opened by default.
- The resolution and controller number are not supported. You can use `DeviceConfig.DEFAULT` for those values. The resolution of ADC is 8 bits.
- The default value for `samplingInterval` is 500000 microseconds (500 ms). The value can be changed immediately during acquisition. This is a platform-specific behavior.
- The sampling time can be also configured. The `samplingTime` value of `DeviceConfig.DEFAULT` is interpreted as 10 usec.

- The maximum possible sampling interval is 5000 milliseconds or 5 sec.

Digital-to-Analog Converter (DAC) Devices

The following Digital-to-Analog (DAC) devices are preconfigured.

Device ID	Device Name	Mapped	Configuration

700	PDM0	Header J5 pin 18	`controllerNumber = DeviceConfig.DEFAULT`
			`channelNumber = 0`
			`resolution = DeviceConfig.DEFAULT`
			`samplingInterval = DeviceConfig.DEFAULT`

701	PDM1	Reserved for future use; not available on Qualcomm IoE	`deviceNumber = DeviceConfig.DEFAULT`
			`channelNumber = 1`
			`resolution = DeviceConfig.DEFAULT`
			`samplingInterval = DeviceConfig.DEFAULT`

Note the following:

- The `channelNumber` parameter can be set to 0, 1, or `DeviceConfig.DEFAULT`, which is interpreted as 0.
- Both the `resolution` and `controllerNumber` values are ignored. You can only use `DeviceConfig.DEFAULT` for those values.
- The default `samplingInterval` value is 500000 microseconds (500 ms). The value can be changed immediately during generation. This is a platform-specific behavior.
- The DAC signal is represented as a PDM (pulse density modulation) signal, so the DAC output value affects only the frequency of output signal, not the voltage level. As such, there is no resolution of the DAC signal in the current implementation; only the `min` and `max` values can be used for calculation of output voltage on the DAC channel.

  For calculation of the output voltage, the following formula can be used: `vOutput = (value * vRef) / (maxValue - minValue + 1)`. Note that `(max - min) == (2^n - 1)` is not applicable.

GPIO Pins

The following GPIO pins are preconfigured.

Device ID	Device Name	Mapped	Configuration

0	GPIO0	Header J5 pin 3	controllerNumber = DeviceConfig.DEFAULT
			pinNumber = 26
			direction = GPIOPinConfig.DIR_INPUT_ONLY
			mode = DeviceConfig.DEFAULT
			trigger = GPIOPinConfig.TRIGGER_RISING_EDGE
			initValue - ignored

1	GPIO1	Header J5 pin 5
		JP7 ADC MUX 0
			controllerNumber = DeviceConfig.DEFAULT
			pinNumber = 25
			direction = GPIOPinConfig.DIR_INPUT_ONLY
			mode = DeviceConfig.DEFAULT
			trigger = GPIOPinConfig.TRIGGER_RISING_EDGE
			initValue - ignored

2	GPIO2	Header J5 pin 7
		JP8 ADC MUX 1
			controllerNumber = DeviceConfig.DEFAULT
			pinNumber = 31
			direction = GPIOPinConfig.DIR_INPUT_ONLY
			mode = DeviceConfig.DEFAULT
			trigger = GPIOPinConfig.TRIGGER_FALLING_EDGE
			initValue - ignored

3	GPIO3	Header J5 pin 9
		JP9 ADC MUX 2
			controllerNumber = DeviceConfig.DEFAULT
			pinNumber = 17
			direction = GPIOPinConfig.DIR_INPUT_ONLY
			mode = DeviceConfig.DEFAULT
			trigger = GPIOPinConfig.TRIGGER_HIGH_LEVEL
			initValue - ignored

4	GPIO4	Header J5 pin 11	controllerNumber = DeviceConfig.DEFAULT
			pinNumber = 32
			direction = GPIOPinConfig.DIR_INPUT_ONLY
			mode = DeviceConfig.DEFAULT
			trigger = GPIOPinConfig.TRIGGER_HIGH_LEVEL
			initValue - ignored

Device ID	Device Name	Mapped	Configuration

5	GPIO5	Header J6 pin 3	`controllerNumber = DeviceConfig.DEFAULT`
			`pinNumber = 28`
			`direction = GPIOPinConfig.DIR_INPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_HIGH_LEVEL`
			`initValue` - ignored

6	GPIO6	Header J6 pin 5
		JP11 pin 2 (to connect to G-sensor interrupt)
			`controllerNumber = DeviceConfig.DEFAULT`
			`pinNumber = 27`
			`direction = GPIOPinConfig.DIR_INPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_RISING_EDGE`
			`initValue` - ignored

7	GPIO7	Header J6 pin 7
		JP12 pin 2 (to connect to light sensor interrupt)
			`controllerNumber = DeviceConfig.DEFAULT`
			`pinNumber = 30`
			`direction = GPIOPinConfig.DIR_INPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_FALLING_EDGE`
			`initValue` - ignored

8	GPIO8	Header J6 pin 7	`controllerNumber = DeviceConfig.DEFAULT`
			`pinNumber = 38`
			`direction = GPIOPinConfig.DIR_INPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_LOW_LEVEL`
			`initValue` - ignored

9	GPIO9	Header J6 pin 11
		JP13 pin 1(to connect to temperature sensor interrupt)
			`controllerNumber = DeviceConfig.DEFAULT`
			`pinNumber = 33`
			`direction = GPIOPinConfig.DIR_INPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_FALLING_EDGE`
			`initValue` - ignored

Device ID	Device Name	Mapped	Configuration

10	GPIO10	Header J7 pin 1
		Relay 1
			controllerNumber = DeviceConfig.DEFAULT
			pinNumber = 18
			direction = GPIOPinConfig.DIR_OUTPUT_ONLY
			mode = DeviceConfig.DEFAULT
			trigger = GPIOPinConfig.TRIGGER_NONE
			initValue = false

11	GPIO11	Header J7 pin 3
		Relay 2
			controllerNumber = DeviceConfig.DEFAULT
			pinNumber = 24
			direction = GPIOPinConfig.DIR_OUTPUT_ONLY
			mode = DeviceConfig.DEFAULT
			trigger = GPIOPinConfig.TRIGGER_NONE
			initValue = false

12	GPIO12	Header J7 pin 5	controllerNumber = DeviceConfig.DEFAULT
			pinNumber = 29
			direction = GPIOPinConfig.DIR_OUTPUT_ONLY
			mode = DeviceConfig.DEFAULT
			trigger = GPIOPinConfig.TRIGGER_NONE
			initValue = false

13	GPIO13	Header J7 pin 7	controllerNumber = DeviceConfig.DEFAULT
			pinNumber = 35
			direction = GPIOPinConfig.DIR_OUTPUT_ONLY
			mode = DeviceConfig.DEFAULT
			trigger = GPIOPinConfig.TRIGGER_NONE
			initValue = false

14	GPIO14	Header J7 pin 9
		Jumper P2 pin 2
		(Used by LED)
			controllerNumber = DeviceConfig.DEFAULT
			pinNumber = 13
			direction = GPIOPinConfig.DIR_OUTPUT_ONLY
			mode = DeviceConfig.DEFAULT
			trigger = GPIOPinConfig.TRIGGER_NONE
			initValue = false

Device ID	Device Name	Mapped	Configuration

15	GPIO15	Header J7 pin 11
		Jumper P3 pin 2
		(Used by LED)
			`controllerNumber = DeviceConfig.DEFAULT`
			`pinNumber = 34`
			`direction = GPIOPinConfig.DIR_OUTPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_NONE`
			`initValue = false`

16	GPIO16	Header J7 pin 13
		Jumper P4 pin 2
		(Used by LED)
			`controllerNumber = DeviceConfig.DEFAULT`
			`pinNumber = 12`
			`direction = GPIOPinConfig.DIR_OUTPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_NONE`
			`initValue = false`

17	GPIO17 or
	LED2
		Header J7 pin 15
		Jumper P5 pin 2
		(Used by LED)
			`controllerNumber = DeviceConfig.DEFAULT`
			`pinNumber = 16`
			`direction = GPIOPinConfig.DIR_OUTPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_NONE`
			`initValue = false`

18	GPIO18	Header J7 pin 17
		Jumper P6 pin 2
		(Used by LED)
			`controllerNumber = DeviceConfig.DEFAULT`
			`pinNumber = 36`
			`direction = GPIOPinConfig.DIR_OUTPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_NONE`
			`initValue = false`

19	GPIO19	Header J7 pin 19	`controllerNumber = DeviceConfig.DEFAULT`
			`pinNumber = 15`
			`direction = GPIOPinConfig.DIR_OUTPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_NONE`
			`initValue = false`

Device ID	Device Name	Mapped	Configuration

20	GPIO20	Header J7 pin 2
		DB9 J12 lower pin 3
			`controllerNumber = DeviceConfig.DEFAULT`
			`pinNumber = 10`
			`direction = GPIOPinConfig.DIR_INPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_LOW_LEVEL`
			`initValue` - ignored

21	GPIO21	Header J7 pin 4
		DB9 J12 lower pin 2
			`controllerNumber = DeviceConfig.DEFAULT`
			`pinNumber = 14`
			`direction = GPIOPinConfig.DIR_INPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_LOW_LEVEL`
			`initValue` - ignored

22	GPIO22	Header J7 pin 6	`controllerNumber = DeviceConfig.DEFAULT`
			`pinNumber = 11`
			`direction = GPIOPinConfig.DIR_OUTPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_NONE`
			`initValue = false`

23	GPIO23	Header J7 pin 8	`controllerNumber = DeviceConfig.DEFAULT`
			`pinNumber = 9`
			`direction = GPIOPinConfig.DIR_OUTPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_NONE`
			`initValue = false`

24	GPIO24	Header J7 pin 10
		DB9 J10 lower pin 2
			`controllerNumber = DeviceConfig.DEFAULT`
			`pinNumber = 37`
			`direction = GPIOPinConfig.DIR_OUTPUT_ONLY`
			`mode = DeviceConfig.DEFAULT`
			`trigger = GPIOPinConfig.TRIGGER_NONE`
			`initValue = false`

Note the following:

- `pinNumber` set to `DeviceConfig.DEFAULT` is interpreted as 34, that is, the GPIO pin connected to the LED using the P3 jumper will be opened by default.

- controllerNumber can be set only to DeviceConfig.DEFAULT, so pinNumber is the unique identifier of the GPIO pin on the Brew MP platform.
- Configuration of the GPIO mode is not supported by the Brew MP platform, so the mode parameter can be set only as DeviceConfig.DEFAULT.
- TRIGGER_BOTH_EDGES and TRIGGER_BOTH_LEVELS are not supported by the Brew MP platform.
- Some GPIO pins are mapped to several physical pins; this allows the programmer to use a GPIO pin in different ways. For example:
  - GPIO1, GPIO2, and GPIO3 can be used to control the ADC multiplexer. See the *Qualcomm IoE Development Platform User Guide* at the following link for more information.

    https://developer.qualcomm.com/mobile-development/development-devices-boards/development-boards/internet-of-everything-development-platform/tools-and-resources
  - GPIO6, GPIO7, and GPIO9 can be used as interrupt pins for the onboard sensors. For more information, see the *Qualcomm IoE Development Platform User Guide* and the sensors data sheet.
  - GPIO10 and GPIO11 can be used to control the state of the on-board relays.
  - GPIO14, GPIO15, GPIO16, GPIO17, and GPIO18 can be used to drive a signal to the on-board LEDs. The same is true for "LEDS" port. For more information, please see the *Qualcomm IoE Development Platform User Guide*.
  - GPIO20, GPIO21, and GPIO24 drive the signal to one of the DB9 connector pins that is available onboard.

GPIO Ports

The following GPIO ports are preconfigured.

Device ID	Device Name	Mapped	Configuration

200	LEDS	Header J7 pin 9 and Jumper P2 pin 2 (used by LED)
		Header J7 pin 11 and Jumper P3 pin 2 (used by LED)
		Header J7 pin 13 and Jumper P4 pin 2 (used by LED)
		Header J7 pin 15 and Jumper P5 pin 2 (used by LED)
		Header J7 pin 17 and Jumper P6 pin 2 (used by LED)
			direction = GPIOPortConfig.DIR_OUTPUT_ONLY
			initValue = false
			pins = 13, 34, 12, 16, 36
			mode of pins = DeviceConfig.DEFAULT
			trigger of pins = GPIOPinConfig.TRIGGER_NONE

Device ID	Device Name	Mapped	Configuration

80	ADC-MUX-SEL	Header J5 pin 5 and JP7 ADC MUX 0
		Header J5 pin 7 and JP8 ADC MUX 1
		Header J5 pin 9 and JP9 ADC MUX 2
			direction = GPIOPortConfig.DIR_ OUTPUT_ONLY
			initValue = false
			pins = 25, 31, 17
			mode of pins = DeviceConfig.DEFAULT
			trigger of pins = GPIOPinConfig.TRIGGER_ NONE

I2C

The following configurations can be used to communicate to I2C slaves.

Device ID	Device Name	Mapped	Configuration

300	G-SENSOR	BMA150	controllerNumber = 1
			address = 56
			addressSize = DeviceConfig.DEFAULT
			clockFrequency = 400000

301	LIGHT-SENSOR	ISL29011	controllerNumber = 1
			address = 68
			addressSize = DeviceConfig.DEFAULT
			clockFrequency = 400000

302	TEMP-SENSOR	ADT7481	controllerNumber = 1
			address = 75
			addressSize = DeviceConfig.DEFAULT
			clockFrequency = 400000

303	BATTERY-GAUGE	On-board battery gauge (specification is unknown)	controllerNumber = 1
			address = 85
			addressSize = DeviceConfig.DEFAULT
			clockFrequency = 400000

Note the following:

- The protocol of the on-board battery gauge is unknown.
- The default clock frequency is 400000 Hz, and is represented by DeviceConfig.DEFAULT. 100000 Hz is also supported as a clockFrequency.
- The I2C device number can be set only to 1, which is also represented by DeviceConfig.DEFAULT.
- addressSize must be set either to DeviceConfig.DEFAULT or to 7. 10-bit addressing mode is not supported.

Pulse Counter

The pulse counter has the following configuration.

Device ID	Device Name	Mapped	Configuration

600	COUNTER	GPIO pin 35, Header J7 pin 7	`controllerNumber = DeviceConfig.DEFAULT`
			`channelNumber = DeviceConfig.DEFAULT`
			`type = TYPE_RISING_EDGE_ONLY`
			`GPIO controllerNumber = DeviceConfig.DEFAULT`
			`GPIO pinNumber = 35`

Note the following:

- Only the values `TYPE_RISING_EDGE_ONLY` and `TYPE_FALLING_EDGE_ONLY` are supported for the `type` parameter on the Brew MP platform.
- The `controllerNumber` and `channelNumber` parameters can be set only to `DeviceConfig.DEFAULT`.
- A minimum supported period is 1000 microseconds (1 millisecond). Values less than 1000 microseconds cause exceptions.

SPI

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

Device ID	Device Name	Mapped	Configuration

400	G-SENSOR	BMA150	`controllerNumber = 1`
			`wordLength = 8`
			`clockFrequency = 26000000`
			`clockMode = 3`
			`address = 0`
			`bitOrdering = DeviceConfig.DEFAULT`

Note the following:

- `DeviceConfig.DEFAULT` passed as `controllerNumber` is interpreted as 1 and because only one SPI device with number 1 is presented on the Qualcomm IoE, only 1 or the DeviceConfig.DEFAULT value of `controllerNumber` is supported.
- `clockFrequency` set to `DeviceConfig.DEFAULT` is interpreted as 2000000 Hz.
- `wordLength` and `bitOrdering` are ignored on the Qualcomm IoE board.
- On the Qualcomm IoE board, address 0 is supported because there is only one CS pin available. The address number can be passed only using the first byte of `address` parameter of `SPIDeviceConfig`.

There are also some global SPI-related options that are set for all SPI slaves:

- Chip select pin mode (`0`: chip select de-assert; `1`: chip select keep asserted). This value is set to `1` by default.
- The minimal frequency value in Hz is set to 0.
- The de-assertion time value is set to 1000 by default.

If you must change any of these properties for some SPI device, you can add the following in the `jwc_properties.ini` file:

```
deviceaccess.spi.{bus_id}.{slave_address}.csMode = {value}
deviceaccess.spi.{bus_id}.{slave_address}.minFreq = {value}
deviceaccess.spi.{bus_id}.{slave_address}.deassertionTime = {value}
```

There are restrictions on Java SPI API usage:

- Only 32 bits per word
- The CS active level cannot be managed with `SPIDevice.begin()` and `SPIDevice.end()methods.`
- The CS cannot be set to `CS_NOT_CONTROLLED` because it is always controlled by the platform driver.

To connect to an external SPI device, remove JP17 jumper. See section 5.2.3, "SPI" in *Qualcomm IoE Development Platform User Guide* for more information.

UART Devices

The following UART devices are preconfigured:

Device ID	Device Name	Mapped	Configuration

40	COM1	DB9 J10 upper port	`controllerNumber =DeviceConfig.DEFAULT`
			`baudRate = 19200`
			`dataBits = DATABITS_8`
			`parity = PARITY_NONE`
			`stopBits = STOPBITS_1`
			`flowcontrol = FLOWCONTROL_NONE`
			`inputBufferSize` - ignored
			`outputBufferSize` - ignored

Note the following:

- There is only one UART port available from the DeviceIO API with ID 40; it has the name `COM1`.
- The `INPUT_DATA_AVAILABLE` and `OUTPUT_BUFFER_EMPTY` events are supported on the Qualcomm IoE board.
- `controllerNumber` can be set only to 1, which is the value presented by `DeviceConfig.DEFAULT`.
- Only the `dataBits` values `DATABITS_5`, `DATABITS_6`, `DATABITS_7`, and `DATABITS_8` are supported.
- `STOPBITS_1`, `STOPBITS_2` are supported
- The following baud rates are supported [bps]: 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800, 500000, 921600, 1000000

Watchdog

The following watchdog devices are preconfigured:

Device ID	Device Name	Mapped	Configuration
500	WDG	Platform Watchdog	N/a
501	WWDG		N/a

Watchdog peripheral with the name WDG is a basic platform WatchdogTimer while with the name WWDG is a WindowedWatchdogTimer.

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.

Modifying 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. Open the `jwc_properties.ini` that is a part of the Oracle Java ME Embedded bundle (or download it from the board using the Brew MP SDK Loader tool), find the property that should be changed, and modify its value.
2. Copy the modified version of the `jwc_properties.ini` file to the `/sys/mod/java` directory on the Qualcomm IoE board using the Brew MP SDK Loader tool.
3. If there is a `jwc_properties.inix` file located in this directory, delete it.
4. Restart Java on the Qualcomm IoE 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.

Using CLI Commands to Alter Network-Related Settings

To alter the network-related settings, do the following:

1. Connect to the board using command-line interface (CLI)
2. Execute a command that starts with the prefix "`net`" to apply a network-related change.
3. Apply the network-related change and restart Java.

Restarting Java on the Qualcomm IoE Board

You can use any of the following methods to restart Java on the Qualcomm IoE board.

1. Use the CLI "`shutdown -r`" command. If the "`vmconfig.reboot_type`" property is set to "`soft`" (the default) and the `reboot_java` application has been deployed on your Qualcomm IoE board, then only Java will be rebooted. Otherwise, if the "`vmconfig.reboot_type`" property is set to "`hard`" or there is no reboot_java application deployed, then the board will be rebooted. Note that the "`vmconfig.reboot_type`" property also affects Device I/O API watchdog's reboot type.
2. Using the CLI "`net-reconnect`" command. This command reconfigures the network and performs a soft Java reboot.
3. Press the "RESET KEY" located on the board, or cycle the power to the board.

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 computing (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, nonprofit group responsible for the standardization of information and communication technologies within Europe. Although based in Europe, ETSI 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.

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 use only 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.

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.

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 the application management system (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 Oracle 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.

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.

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 runs 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 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 Aescriptor 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 cell 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

A process of verifying Java technology classes. 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 accomplishment of individual instructions. The ARM architecture is based on RISC design principles.

RL-ARM

Real-Time Library ARM. 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 multitasking, 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 card. A nonvolatile memory card format for use in portable devices, such as cell 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 SMS 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), then 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. 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 cell 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.