ORACLE

Java™ Engineering Services

Java ME User Guide for Qualcomm IoE Development Platform (QSC6270T)

Jul 4, 2013

Version 1.0

Table of Contents

1.Introduction

This document describes the contents of Java ME Embedded binary for the Qualcomm IoE Development Platform (QSC6270T). It also describes how to install, run, and debug Oracle Java ME Embedded software on the the Qualcomm development board.

The scope of the Java ME Embedded binary covers the IMP-NG and Headless AMS libraries, including the following JSRs:

- JSR 75 (FC)
- JSR 120 (WMA)
- JSR 139 (CLDC 1.1)
- JSR 172 (Web Services)
- JSR 177 (SATSA – CRYPTO only)
- JSR 179 (Location API)
- JSR 228 (IMP-NG)
- JSR 280 (XML API)

The Java ME Embedded binary also contains the following features:

- *Headless AMS & CLI*—Programmers can connect to the device both through Universal Asynchronous Receiver/Transmitter (UART) ports or TCP/IP network connections to open a command line interface (CLI). This CLI provides various application management services such as installing, running, and removing Java ME applications.

- *Device Access APIs*—The Device Access APIs allow connections to various peripherals connected to the board through a variety of buses, such as General Purpose Input/Output (GPIO), Inter-Integrated Circuit (I^2C), Serial Peripheral Interface (SPI), Analog-to-Digital Conversion (ADC), and AT Commands.

Most of the features in this binary are based on the Oracle Java ME Embedded 3.2 release. ADC, AT-commands and Watchdog APIs are newly added. The Java ME SDK Developer's Guide, version 3.2, which is available at http://docs.oracle.com/javame/developer/developer.html, should be the primary reference when developing Java embedded applications. This document - 'Java ME User Guide for Qualcomm IoE Development platform (QSC6270T)' - is a supplement which tries to address items specific to the Qualcomm development board which is beyond the scope of the Java ME SDK Developer's Guide.

1.1. Reference Documents:

Document Name	Document ID	Version	Date
Oracle® Java ME Software Development Kit Developer's Guide Release 3.2 for Windows	E24265-04	3.2	September 2012
Oracle® Java ME Software Development Kit Developer's Guide Release 3.2 for Eclipse Windows	E37550-02	3.2	September 2012
Oracle® Java ME Software Development Kit Release Notes Release 3.2 for Windows	E25309-06	3.2	October 2012

1.2. Definitions and Acronyms:

CLDC HI	Connected, Limited Device Configuration – Hotspot Implementation
MIDP	Mobile Information Device Profile
IMP-NG	Information Module Profile – Next Generation (Headless subset of MIDP 2.0)
IMlet	Java application build to run on the IMP-NG profile
JSR	Java Specification Request
ODD	On-Device Debugging
AP	Access Point
CLI	Command Line Interface (reference interface for the Headless AMS)

2. Contents of the Java Runtime Binary

The Oracle Java ME Embedded runtime binary consists of the following key files.

Files	*Destination*	*Description*	*New or changed ?*
_function_groups.txt	fs:/sys/mod/java/appdb	Security function group file	*
_main.ks	fs:/sys/mod/java/appdb	Security key store file	*
_policy.txt	fs:/sys/mod/java/appdb	Security policy file (reference with limited permission)	*
_policy.dev.txt	fs:/sys/mod/java/appdb	Security policy file for development (allows all permission)	*
jwc_properties.ini	fs:/sys/mod/java	Default property settings	*
jwc_properties.reference.ini	N/A	Reference file with default property settings with comments and spacing for editing manually	*
java.mif	fs:/sys/mod/java	BREW module information file	*
java.mod	fs:/sys/mod/java	Java ME binary release for the BREWMP device	*
watchdog.ini	fs:/sys/mod/java	Property file for keeping internal counter info.	*

3. Using the Oracle Java ME Embedded Runtime

The following steps show how to deploy and run the Oracle Java ME Embedded runtime on the Qualcomm Orion board. For basic instructions on configuring and powering on the hardware, you should refer to the Qualcomm document.

3.1. Deploying Oracle Java ME Embedded Binaries to the Development Board

NOTE: This step is only need if no pre-existing Java runtime is on the development board or if you need to upgrade the existing Java runtime binaries. You should first refer to Qualcomm documents for installing USB drivers for the development board and obtaining tools needed.

First, copy the Java runtime binaries to the development board. Copy all files under 'java' directory to the device at 'fs:/sys/mod/java'. The directory structure should be preserved on the device. Make sure there is only single instance of the binary among all 'mod' directories and another copy does not exists in other directories, including 'fs:/usermods/java' and 'fs:/mod/java'.
There is also another module that co-exists with the Java runtime binaries. Copy all files under 'netsetup' directory to the device at 'fs:/sys/mod/netsetup'. This module is automatically launched after device bootup to configure the network and then start Java.

3.2. Starting Up the Java Runtime

The next step is to reboot the device. The Oracle Java ME Embedded runtime is configured to start automatically on system bootup. Depending on the configuration, it may take some time after the power on for Java to start running. Usually configuration of cellular network and/or connecting to configure wireless AP occurs before Java is run. After a successful start up, Java enables the Command Line Interface (CLI) via the Virtual COM port over USB.

3.3. Verifying if Java is running properly

The next step is to verify that the the Oracle Java ME Embedded runtime is running correctly. The easiest way is to verify bootup logging messages and verifying if the CLI is working correctly. By default, CLI is configured to output logs to the Virtual COM over USB. You can view the bootup log and use the CLI with any terminal applications that can communicate using a COM port on the PC. Please refer to details for configuring the device and termial applications for CLI under 4.4 Using Serial CLI over USB or UART.

4. Java Command Line Interface (CLI)

The Oracle Java ME Embedded CLI is a simple command line for the headless Application Management System (AMS). It allows the programmer to perform basic application management such as installing and running applications, as shown in Table 4.1. Currently by default, CLI is available through the Virtual COM port over USB.

To install application using HTTP, the device needs access to a HTTP server hosting the installable applications either through WiFi or cellular network connection.

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

4.1. Basic CLI Command List

Syntax	Description
ams-list [INDEX or NAME|VENDOR]	List all installed IMlet suites and their statuses or show the detail of a single suite
ams-install <URL>	Install an IMlet using the specified JAR or JAD file, specified as a URL
ams-run <INDEX or NAME|VENDOR>	Execute the specified IMlet
ams-remove <INDEX or NAME|VENDOR>	Remove an installed IMlet
ams-update <INDEX or NAME|VENDOR>	Update the installed IMlet
ams-stop <INDEX or NAME|VENDOR>	Stop the specified IMlet
ams-suspend <INDEX or NAME|VENDOR>	Suspend (pause) the specified IMlet
ams-resume <INDEX or NAME|VENDOR>	Resume the specified IMlet
ams-info <INDEX or NAME|VENDOR>	Show information about the installed IMlet
help [command name]	List the available commands or detailed usages for a single command
sysmenu <on|off>	Enable hidden system menu commands

* Note: Advanced system menu commands list can be found at 4.6System Menu Commands.

* Note that "`Name|Vendor`" can be used to specify an IMlet. For example, if the name is "`Helloworld`" and vendor is "`vendor`", you can type the following:

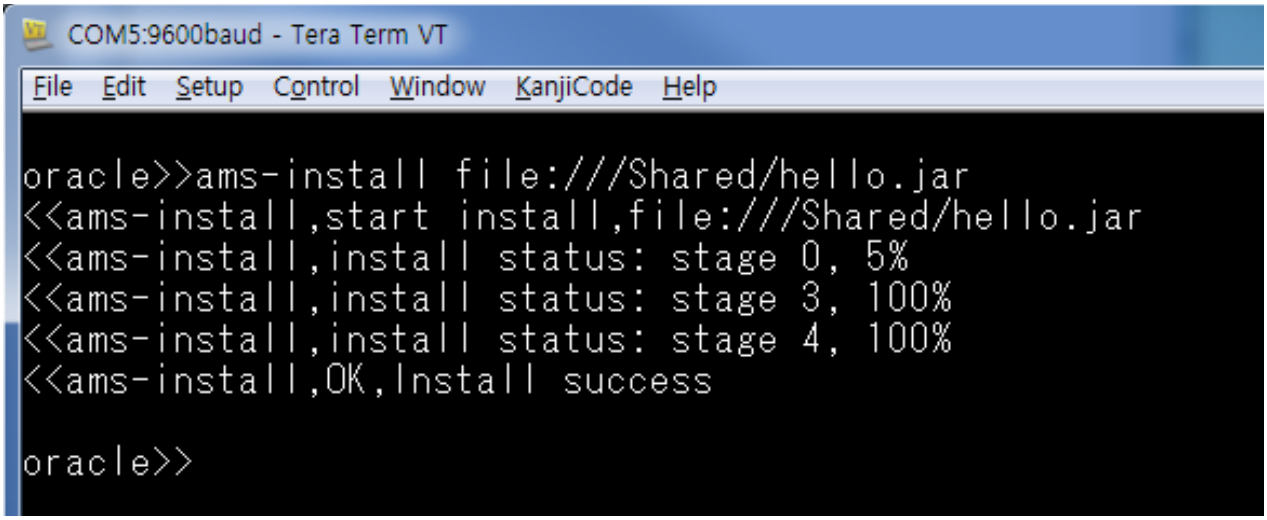
```
oracle>> ams-remove Helloworld|vendor
```

It is often simpler to use the integer index of the application shown via 'ams-list' command.

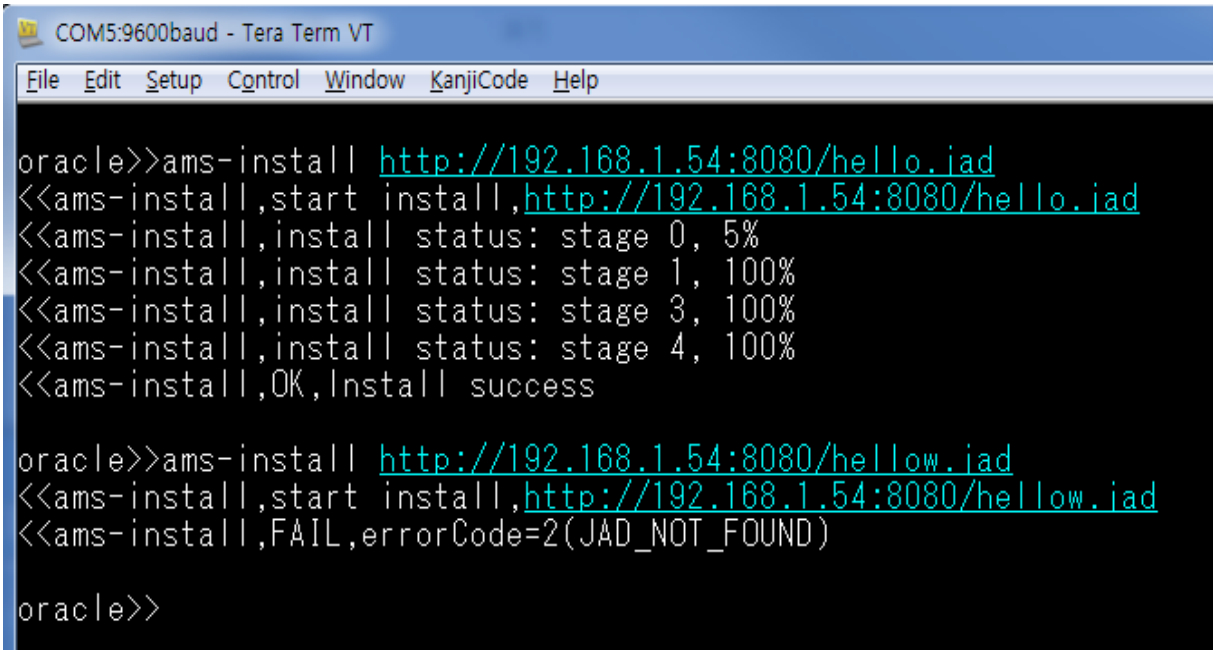
Here is a typical example of using the Application Management System (AMS) to install, list, run, and remove a Java ME Embedded application on the board.

The following screenshots demonstrate using TeraTerm to make a CLI connection to the Qualcomm Orion board.

the example of IMlet installation from the "Shared" folder

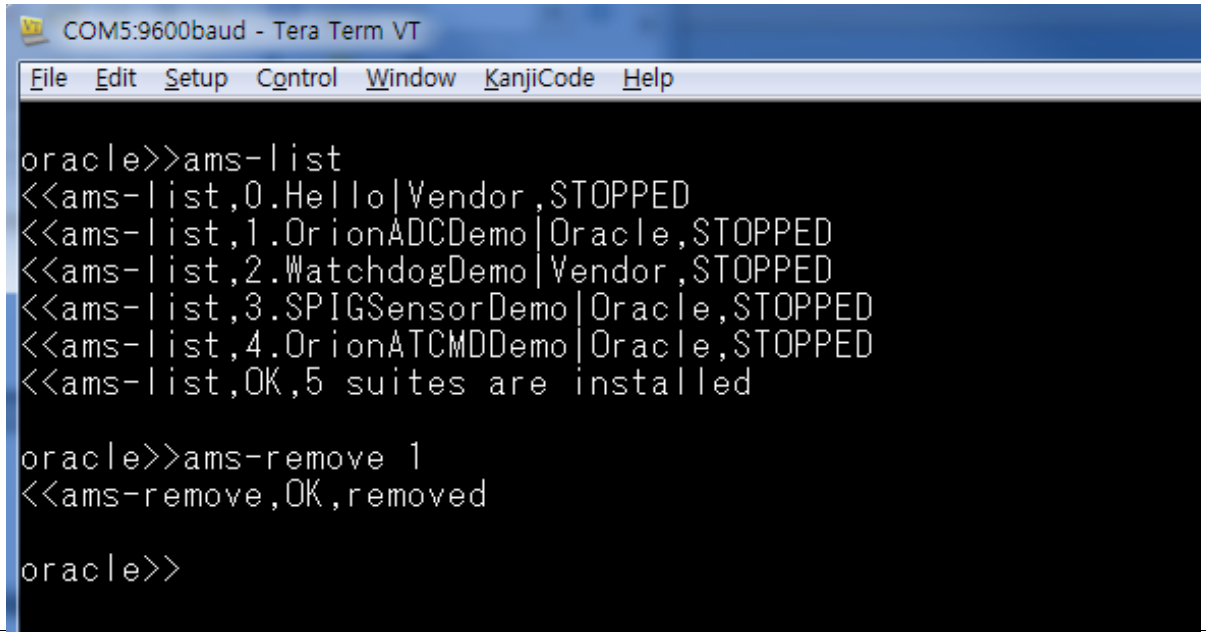


the example of IMlet installation from HTTP server

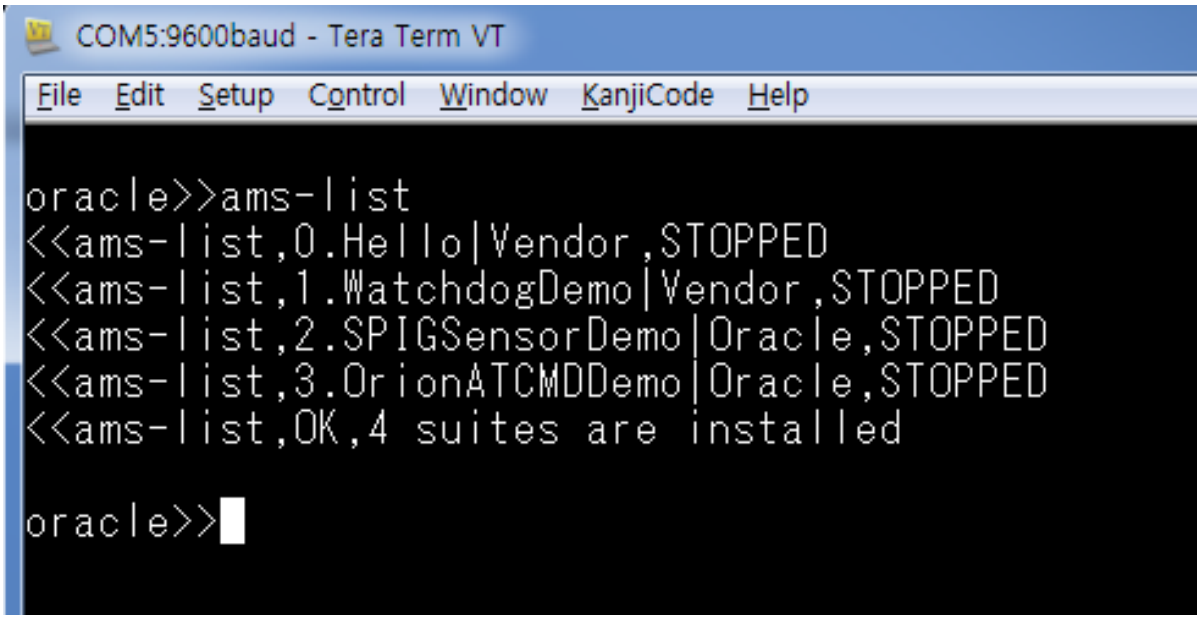


*Note that the final installation example failed with an error code and matching description. If the install process shows any error code, see D, "AMS Installer Error Codes" for more information on how to resolve the error.

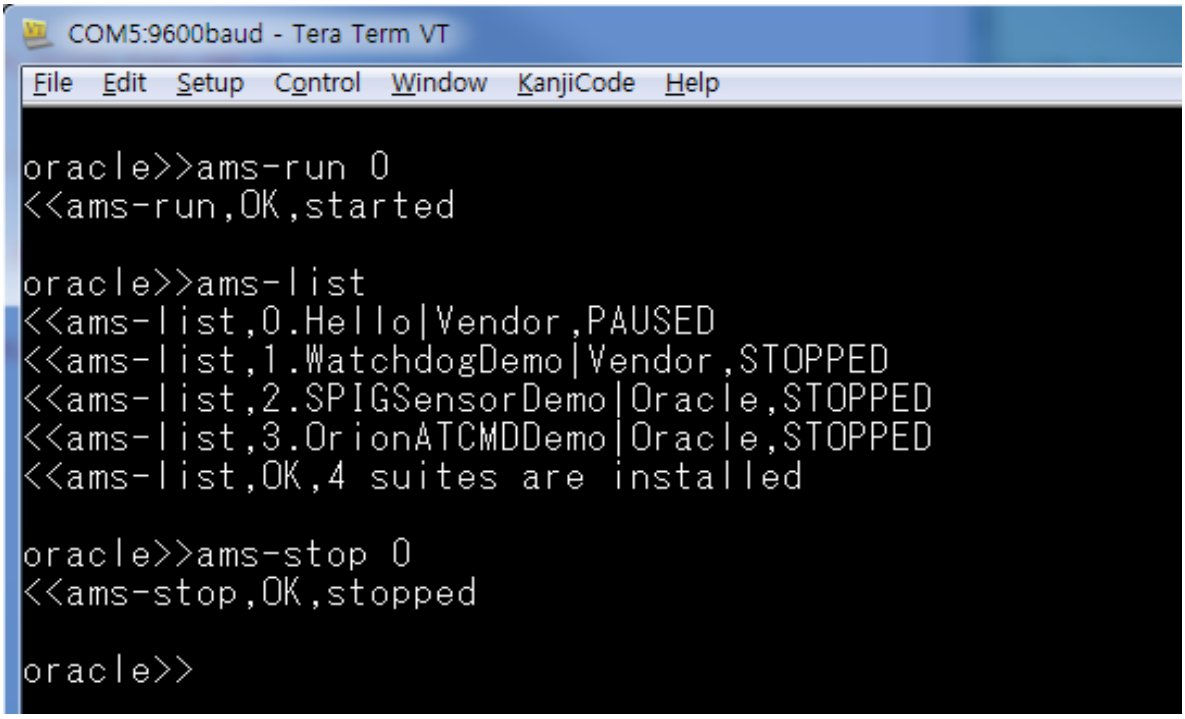
Once an IMlet is installed, verify it using the ams-list command. Here, we have



4 other IMlets other than Hello IMlet. Each one has been assigned a number by the AMS for convenience.And the ams-remove command can be used to remove any installed IMlet.
The results can again be verified with the `ams-list` command.



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



4.2. Options for Installing Java Applications

The command-line must be used to install IMlets. Note that the CLI is only used for issuing commands and application installation itself requires a separate channel, such as an SD card or using HTTP.

If you develop application using Java ME SDK on NetBeans or Eclipse, installing and running applications on the development board is done automatically when doing run or debug with an application. For further instructions see chapter 7 on device debugging.

4.2.1. Installing Java Application via HTTP

To install a Java ME application across the Internet, upload the application JAD and JAR files to a public web server that the development board can access via HTTP (needs WiFi or GSM/WCDMA). Then, install the application using the CLI command 'ams-install' specifying the appropriate URL:

```
ams-install http://192.168.0.6/AppName.jad
```

4.2.2. Installing Java Application from a SD Card

To install a Java ME Application from an SD card (fs:/card0), first, copy the JAD and JAR files to an SD card. Next, insert the SD card into the memory card slot on the development board, and install using the CLI command '`ams-install`' with path of the memory card:

```
ams-install file:///MemoryCard/AppName.jad
```

4.2.3. Installing Java Application from a device local file system

If you can access the device file system using 'Loader' tool in Brew MP SDK, you can copy the JAD and JAR files to shared folder of the device (fs:/shared). Then, install using the CLI command '`ams-install`' with path to the shared directory:

```
ams-install file:///Shared/AppName.jad
```

* Note: '`Shared`' should contain capital '`S`'.

4.3. Choosing CLI Connection Type

The CLI can be configured to use either a TCP/IP connection over WiFi, or a serial connection using the Virtual COM port over USB or the on-board serial port. By default, it is configured to use the Virtual COM port over USB.

The following properties defined in 'jwc_properties.ini' can be used to configure the CLI connection mode. You should be able to set or get the properties through the '`setprop`' and '`getprop`' commands. See chapter Editing Runtime Properties for more information.

Note: 'SER3' and 'COM1' are names used on the device. Matching COM port on the PC side will likely be a different number.

Virtual COM over USB (default)

```
com.oracle.midp.ams.headless.cli.connectiontype = 1
com.oracle.midp.ams.headless.cli.comname = SER3
com.oracle.midp.ams.headless.cli.baudrate = 115200
```

On-board Serial Port (UART)

```
com.oracle.midp.ams.headless.cli.connectiontype = 1
com.oracle.midp.ams.headless.cli.comname = COM1
com.oracle.midp.ams.headless.cli.baudrate = 115200
```

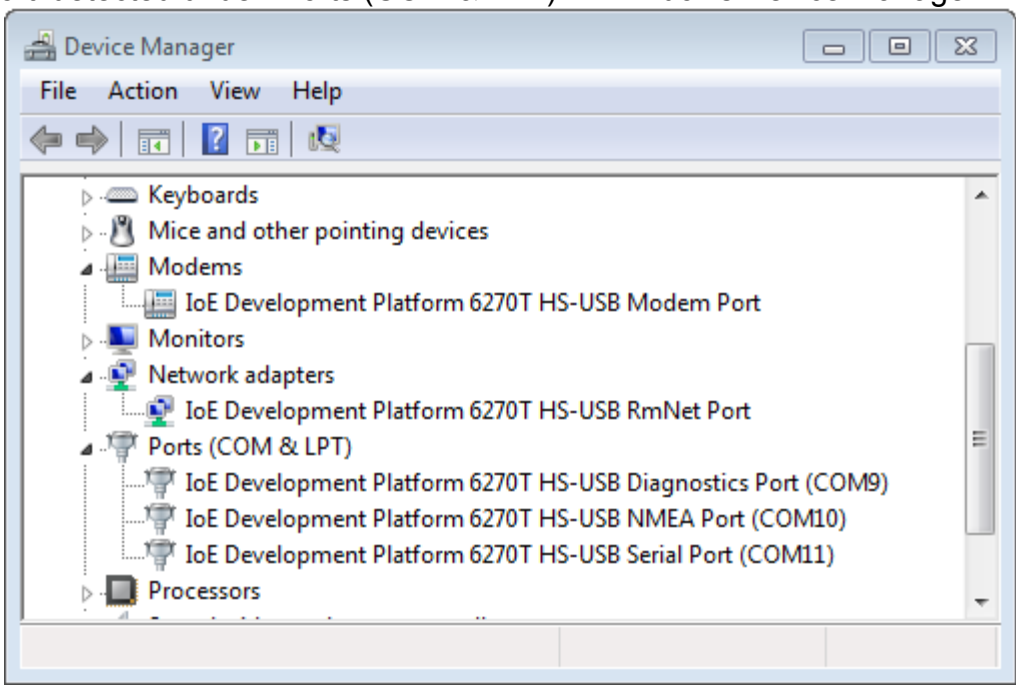
TCP/IP (via WiFi)

```
com.oracle.midp.ams.headless.cli.connectiontype = 0
system.wlan.ap.ssid = myWiFiAP
system.wlan.ap.passwd = ifneeded
```

4.4. Using Serial CLI over USB or UART

- Using a serial connection is the preferred way to use the CLI. With serial connection you get additional logging during bootup and also during Wireless LAN connection. Also it does not depend on external Wireless AP so it is guaranteed to work every time.
- Make sure USB or serial cable is connected between the device and the PC. Also for Virtual COM over USB, drivers distributed with the board should be already installed.

- After device bootup you should see ‘IoE Development Platform * HS-USB Serial Port’ detected under ‘Ports (COM & LPT)’ in Windows Device Manager.



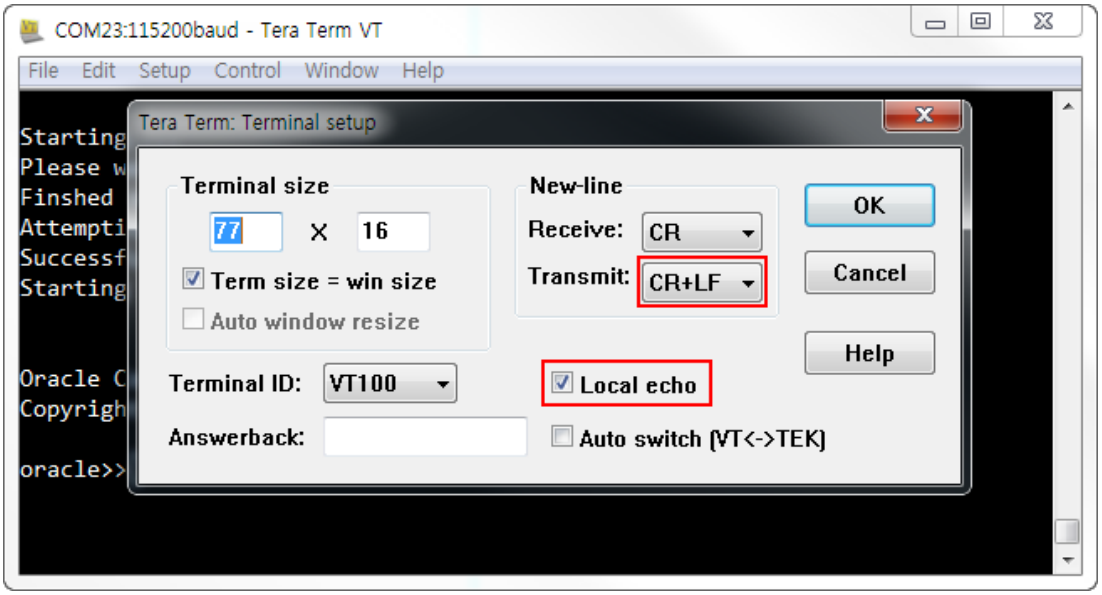
- When the device boots up and Java is running, you can connect to the CLI using this Virtual COM port.

- For connecting to the on-board UART, choose the proper port supported by your PC.

- Use a terminal application on the PC side to view messages from the CLI and send commands. The PC terminal should use the following values for the serial port settings:
  - Baud rate: 115200bps
  - Data: 8 bit
  - Parity: None
  - Stop: 1bit
  - Flow control: None

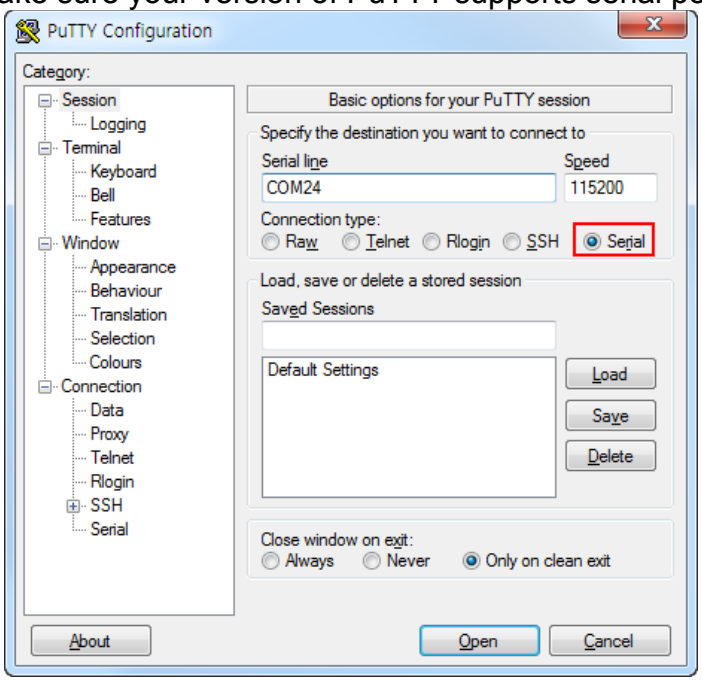
- Additionally following terminal option needs to be set as well:
  - Enable ‘local echo’
  - Transmit LF(Line Feed) along with with CR (Carriage Return)
    - This should be for the outgoing data not incoming

- Below are configuration info for some of popular terminal applications
  - Configuration for Tera Term
    - From menu ‘Setup’ > ‘Terminal’, select ‘CR+LF’ for Transmit and check ‘Local Echo’

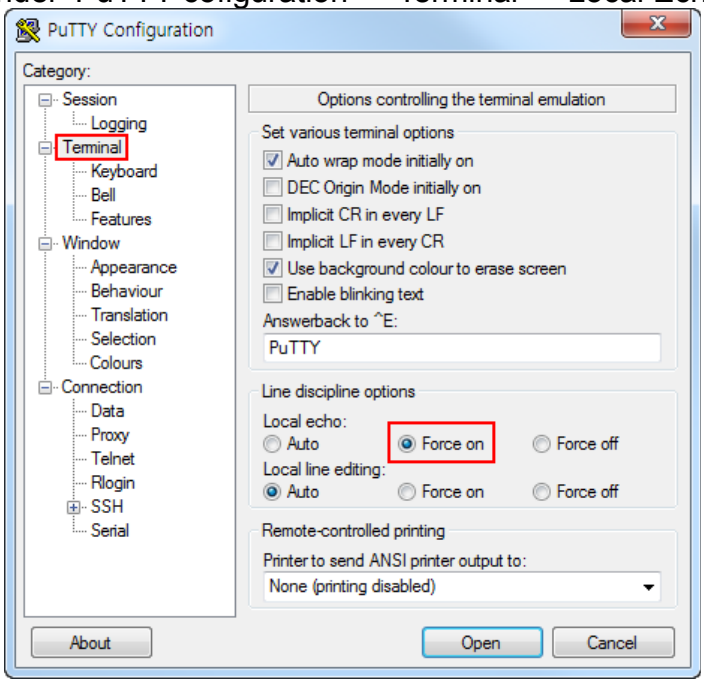


CAUTION: when rebooting the device, you should always disconnect the Virtual COM connection. Some times leaving this open prevents the device from booting up. This does not apply to the on-board UART connection.

- Configuration for Putty
  - Make sure your version of PuTTY supports serial port connection



  - Under ‘PuTTY cofiguration’ > ‘Terminal’ > ‘Local Echo’ > select ‘Force On’



- After the terminal is configured correctly, power on or reset the device. After Java starts, CLI will send a custom prompt message to the client terminal. If the terminal client on PC shows the following message and prompt than Java is running successfully

```
Oracle Command Line Interface  (build: Apr 12 2013 08:18:24)
Copyright (c) 2012, 2013, Oracle

oracle>>
```

4.5. Using TCP/IP CLI over WiFi

Although it is preferable to use the serial CLI, you may need to configure the CLI for TCP/IP over WiFi connection sometimes. You can use the initial serial CLI connection to setup to connect to a wireless AP. The development board will use the WiFi configuration to connect to the wireless AP and automatically obtain an IP address using DHCP. Your PC need to be able to access this IP address to use TCP/IP CLI or on device debugging (ODD) via NetBeans or Eclipse. When using TCP/IP CLI, you can do a telnet to the ORION device to access the CLI.

- Telnet IP address : IP address of the development board
- Telnet Port : default 65002. Can be changed with '`setprop`' CLI command

```
>> setprop com.oracle.midp.ams.headless.listening.port 12345
>> saveprops
```

- You need to view the Java runtime logs to view the obtained IP address and port
  - You can use the 'Logger' tool which included in BrewMP SDK to view the logs.
  - Look for the following logs:

```
[CLI]TCP start listen: 192.168.1.52:65002
[CLI]waiting client
```

  - If such logs are not found it is likely a problem has occurred. Look for the following logs to get hint for debugging the problem. See Error Messages shown during Network Setup for more details about error messages.

```
[NetSetup] WLAN ERROR: ...
```

- Alternatively using the fact that IP address obtained from a wireless AP usually does not change frequently, you can connected to a wireless AP using the serial CLI and use the 'net info' command to obtain the device IP address. Later try connecting to the TCP/IP CLI using the previously obtained IP address. Note this is not guaranteed to work always.

4.6. System Menu Commands

When the **sysmenu** command is entered with the **on** option, additional system menu commands are available with the AMS CLI, as shown below.

```
>> sysmenu on
```

Commands for editing Java runtime properties

The following commands are used for reading and editing Java runtime properties. This has the same effect of editing 'jwc_properties.ini' file manually if you have access to the device file system.

Syntax	Description
getprop <KEY>	for viewing internal properties defined in 'jwc_properties.ini'
setprop <KEY> <VALUE>	for editing or adding internal properties defined in 'jwc_properties.ini'. Need to call 'saveprops' to make the change permanent. Only the properties listed below can be edited or added. (* is wildcard). Also edited properties is kept on the RAM and not written to the 'jwc_properties.ini' file until 'saveprops' is called. ◦ deviceaccess.* ◦ com.oracle.midp.ams.headless.* ◦ sms.port.global ◦ system.wlan.* ◦ system.network.* ◦ system.netsetup.timeout ◦ system.java.init.delay
saveprops	for saving the edited properties to the 'jwc_properties.ini' file. This should be called to make any changes using 'setprop' permanent.

Configuration commands for network setup

The following commands are also available to configure the device or WiFi or cellular network.

Command	Usage	Description
net info	net info	displays IP, SSID, AuthType, signal strength
net set ssid	net set ssid [SSID]	sets the SSID string of the wireless AP
net set passwd	net set passwd [PASSWORD]	sets the 64 digit hexadecimal passkey for connection
net set pref	net set pref [0|1|2|3|4|5]	Mode preference value – 0:AUTO, 1:NO OP, 2:WLAN Only, 3:GSM/WCDMA only, 4:WCDMA only, 5:GSM/WCDMA/WLAN
net set apn	net set apn [APN]	sets the APN string
net set pdp_authtype	net set pdp_authtype [0|1|2]	Sets the PDP authentication type
net set pdp_username	net set pdp_username [USERNAME]	Sets the PDP username
net set pdp_password	net set pdp_password [PASSWORD]	Sets the PDP password
reboot	reboot	Reboots the device (safer way to reboot compared to using the H/W reset button)

- All '`net set`' commands saves the value directly to 'jwc_properties.ini' file without needing to run the '`saveprops`' command
- After setting the network properties you need to reset the device for this to take effect.
  - After resetting the device, if you have a successful CLI connection '`net info`' should show a valid IP address.
  - IP of 0.0.0.0 means that connection was not successful.
    - For WiFi connection, it will remain unconnected until device is reset
    - For cellular connection, if all the settings are proper IP address may be obtained later depending on the cellular network status

4.7. Editing Runtime Properties

The '**jwc_properties.ini**' file contains basic configuration properties used by the Java runtime. It is read on JVM startup. There are only a few configuration values that you need be edit to run Java successfully. Most values should be left at their default value. You can either edit the file directly and copy it over to the device file system, or for some of the values you can use the CLI '`getprop`' and '`setprop`' command to read or edit the existing values. Note that for network connection using WiFi, you must change the default value of the SSID.

Key	Value	Description
system.wlan.ap.ssid	SSID string	SSID of wireless AP
system.wlan.ap.passwd	password string	Password of wireless AP (WPA/WPA2 -PSK only). Ignored on open AP.
com.oracle.midp.ams.headless. listening.port	Port number	Port number of CLI when using TCP connection (default: 65002)
system.netsetup.timeout	Time in milliseconds	Network setup should run after bootup. Java will automatically start if the process is not finished within timeout (default: 45000)
system.java.init.delay	Time in milliseconds to wait before launching Java	Some delay is given to allow for DHCP to safely complete after WLAN_CONNECTED event (default: 7000)
com.oracle.midp.ams.headless.cli. connectiontype	0 or 1	CLI connection type: 0 = TCP, 1 = Serial
com.oracle.midp.ams.headless.cli. comname	COM1 or SER3	COM1 = UART, SER3 = Virtual COM (USB)
com.oracle.midp.ams.headless.cli. baudrate	bps value	Baud rate of Serial port (default: 115200)
sms.port.global	Integer value	Special port where all SMS without port information is forwarded to (default: 65533)

5. Viewing Java Runtime Logs

There are two types of logs that can be viewed. One is the logs printed by the Java application and this can be viewed within NetBeans or Eclipse. Another option is viewing the logs from the Java runtime which also includes logs from the application. Unless you are a system integrator or trying to debug low level issues with the underlying platform or the Java runtime, viewing Java runtime logs is usually not needed.

- To view Java runtime logs, you need to use the Logger tool from the Brew MP SDK.
- You can use a TCP/IP connection to view the application logs as well. This is not recommended but works as a temporary solution when NetBeans or Eclipse debugging is not used or available. It is done by connecting to the `stdout` port used for on device debugging. You can do a telnet to the device port 51235 using Windows telnet client. However this should not be used together when doing on device debugging as this can mess up the debugging session.

6. Device Access APIs

Java runtime for the Qualcomm IoE Development Platform is a custom runtime created based on Java ME Embedded product version 3.2 with additional device access APIs not available in the 3.2 release. The Java runtime is being upgraded to 3.3 but in the meanwhile developers need to take into account the version differences.

What this means for the developers are:

- Use Java ME SDK 3.2 SDK and plug-ins instead of more recent versions
- Include additional library to you project for building ADC, AT commands and watchdog APIs as it is not included in the 3.2 SDK.

Following are the APIs that are already available in Java ME Embedded 3.2:

- General Purpose Input Output (GPIO)
- Inter-Integrated Circuit (I2C)
- Serial Peripheral Interface (SPI)

To use these APIs you can reference the existing documentation found at the following link: http://docs.oracle.com/javame/config/cldc/opt-pkgs/api/daapi/index.html

Following are the API that are available for Java ME Embedded 3.2 but added for the Qualcomm IoE Development Platform:

- Analog to Digital Converter (ADC)
- AT commands
- Watchdog

To use these APIs you can reference the javadoc files that is distributed with the Qualcomm IoE development Platform. These APIs were adopted from 3.3 to fit the API usage for 3.2. All above APIs (ADC,AT Command and Watchdog) needs slight modification to run on the 3.3 binary and the migration guide can be found at the following link:

http://docs.oracle.com/javame/config/cldc/rel/3.3/core/da/html/device_access/html/device_access/appendix.htm

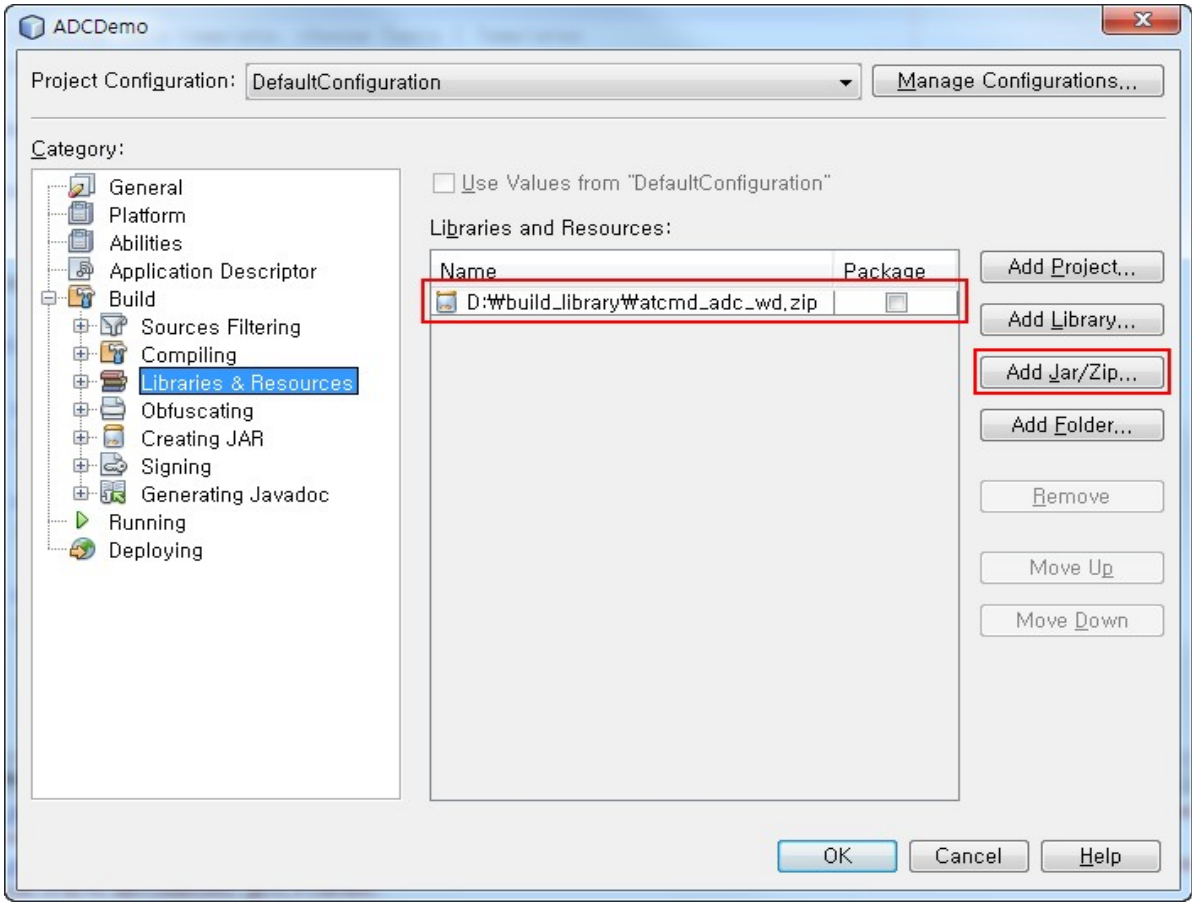
6.1. Building Projects using ADC, AT commands or Watchdog APIs

As these APIs are not supported by the 3.2 SDK so you need to add a library defining these APIs to your project to successfully the build it. If you do not include this library in the project you may get build errors saying the package does not exist.

```
error: package com.oracle.deviceaccess.atcmd does not exist
error: package com.oracle.deviceaccess.adc does not exist
error: package com.oracle.deviceaccess.watchdog does not exist
```

Adding the library for building can be done using the following steps :

- In your NetBeans project, open project properties windows.
- Under Category, select 'Build > Libraries & Resources'.
- Click on 'Add Jar/Zip...' Button
- From the file selection dialog, locate and open 'atcmd_adc_wd.jar' file that came with the Java User Guide.
- Make sure you uncheck the 'Package' checkbox. This library does not need to be packaged since the device Java runtime already includes the API.



6.2. Limitations of AT commands API

Note that some of the Oracle Java ME Embedded APIs are not supported do to missing support in the underlying system APIs. As such, any use of the following APIs will throw an UnsupportedOperationException.

- ATDevice.abortCommand()
- ATDevice.escapeToCommandMode()
- ATDevice.isConnected()
- ATDevice.openDataConnection()
- ATDevice.openDataConnection()
- DataConnection.*
- DataConnectionHandler.*

6.3. Accessing External Peripherals from Java

You can attach your own peripherals to the Qualcomm development board. For the hardware wiring, please view the appropriate Qualcomm hardware document for further information.

To access the attached external peripheral from Java, configuration information needs to be available to the Java runtime. Currently peripheral configuration info is contained in the ‘jwc_properties.ini’ file. New properties can be added by issuing ‘`setprop`’ system menu command from the CLI, or by adding entries directly to the 'jwc_properties.ini' file.

Custom peripherals interfaces that can be wired to the development board and can be used by Java is GPIO, I2C, SPI and ADC.

6.3.1. Property values for Peripherals

You can refer to the following property info for on-board peripherals to add additional custom peripheral configuration information. Normally you need to increment the count of the peripheral or bus you are trying to add and add peripheral properties using index of ‘count – 1’. It works similar to array length and array indexing.

Peripheral Type	Property Example

I2C	# The number of I2C slaves that are available
	deviceaccess.i2c.slaveCount = 4
	# I2C slave's name
	deviceaccess.i2c.slave0.name = G-SENSOR
	# I2C slave's ID
	deviceaccess.i2c.slave0.id = 300
	# I2C bus ID of the I2C slave
	deviceaccess.i2c.slave0.busId = 1
	# I2C slave's address
	deviceaccess.i2c.slave0.address = 56

SPI	# The available slave number of the SPI
	deviceaccess.spi.slaveCount = 1
	# ID of slave
	deviceaccess.spi.slave0.id = 400
	# name of slave
	deviceaccess.spi.slave0.name = G-SENSOR
	# The number of bit of slave's word.
	deviceaccess.spi.slave0.wordSize = 8
	# minimum frequency of slave
	deviceaccess.spi.slave0.minFreq = 0
	# maximum frequency of slave(Hz)
	deviceaccess.spi.slave0.maxFreq = 26000000
	# 0:chip select de-assert , 1:chip select keep asserted
	deviceaccess.spi.slave0.csMode = 1
	# 0:active low 1:active high

	deviceaccess.spi.slave0.csPolarity = 0 # 0:clock idle low 1:clock idle high deviceaccess.spi.slave0.clkPolarity = 1 # 0:input first mode 1:output first mode deviceaccess.spi.slave0.transferMode = 1

GPIO Pin	# The number of gpio pins that are available
	deviceaccess.gpio.pinCount = 25
	# GPIO pin's ID.
	deviceaccess.gpio.pin0.id = 0
	# GPIO pin's name.
	deviceaccess.gpio.pin0.name = GPIO0
	# GPIO pin's output mode. 1:output 0:input
	deviceaccess.gpio.pin0.direction = 1
	# GPIO pin's initial value.
	deviceaccess.gpio.pin0.initialValue = 0
	# polarity 0:active low 1:active high
	deviceaccess.gpio.pin0.polarity = 0
	# detection type 0:level 1:edge
	deviceaccess.gpio.pin0.detection = 0
	# GPIO pin number on IoE module
	deviceaccess.gpio.pin0.number = 26

GPIO Port	# The number of gpip ports that are available
	deviceaccess.gpio.portCount = 1
	# GPIO port's output mode. 1:output 0:input
	deviceaccess.gpio.port0.direction = 1
	# GPIO port number
	deviceaccess.gpio.port0.number = 0
	# GPIO port's max value size
	deviceaccess.gpio.port0.maxVal = 3
	# GPIO port's name
	deviceaccess.gpio.port0.name = LEDS
	# GPIO port's ID
	deviceaccess.gpio.port0.id = 1
	# GPIO port's initial value
	deviceaccess.gpio.port0.initialValue = 0
	# GPIO port's pin number
	deviceaccess.gpio.port0.pinCount = 2
	# GPIO port's pin number
	deviceaccess.gpio.port0.pin0.number = 16
	# GPIO port's pin number

	deviceaccess.gpio.port0.pin1.number = 17# The number of gpio ports that are available
	deviceaccess.gpio.portCount = 2
	# GPIO port's ID
	deviceaccess.gpio.port0.id = 80
	# GPIO port's name
	deviceaccess.gpio.port0.name = ADC-MUX-SEL
	# GPIO port's output mode. 1:output 0:input
	deviceaccess.gpio.port0.direction = 1
	# GPIO port number
	deviceaccess.gpio.port0.number = 0
	# GPIO port's max value
	deviceaccess.gpio.port0.maxVal = 7
	# GPIO port's initial value
	deviceaccess.gpio.port0.initialValue = 0
	# GPIO port's pinCount
	deviceaccess.gpio.port0.pinCount = 3
	# GPIO port's pin number
	deviceaccess.gpio.port0.pin0.number = 25
	# GPIO port's pin number
	deviceaccess.gpio.port0.pin1.number = 31
	# GPIO port's pin number
	deviceaccess.gpio.port0.pin2.number = 17

ADC	# The available channel number of the ADC
	deviceaccess.adc.channelCount = 2
	# Name of channel
	deviceaccess.adc.channel0.name = VTHERM_N
	# ID of channel
	deviceaccess.adc.channel0.id = 100
	# setting key of channel
	deviceaccess.adc.channel0.settingKey = /BREW/HKADC/VTHERM_N/ENABLE
	# value key of channel
	deviceaccess.adc.channel0.valueKey =

```
/BREW/HKADC/VTHERM_N/VALUE
# sampling interval of channel in msec
deviceaccess.adc.channel0.samplingInterval = 500
# maximum sampling interval of channel in msec
deviceaccess.adc.channel0.maxSamplingInterval = 5000
# minimum sampling interval of channel in msec
deviceaccess.adc.channel0.minSamplingInterval = 100
# maximum value of this channel
deviceaccess.adc.channel0.maxVal = 4095
# minimum value of this channel
deviceaccess.adc.channel0.minVal = 0
# Reference Voltage of this ADC channel(2.2V = 2200)
deviceaccess.adc.channel0.refVol = 2200
```

7. On-Device Debugging (ODD)

It is possible to run or debug Java applications on the Qualcomm Orion board using NetBeans or Eclipse IDE. This process is called On-Device Debugging (ODD). First you have to have proper version of NetBeans or Eclipse installed along with the Java ME SDK 3.2. We recommend Netbeans 7.2.1 for ODD due to be optimized with OJWC 3.2. More information on the IDE and SDK can be found at the following link: http://docs.oracle.com/javame/dev-tools/jme-sdk-3.2/release-notes/toc.htm.

Netbeans 7.2.1 is recommended IDE for Java ME SDK 3.2

It is assumed that you have installed required tools such as plugin for NetBeans and OJWC 3.2 SDK following the above link. Please refer to the “How to install” section of OJWC 3.2 SDK web page :

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

Detailed steps for debugging on the device are covered below.

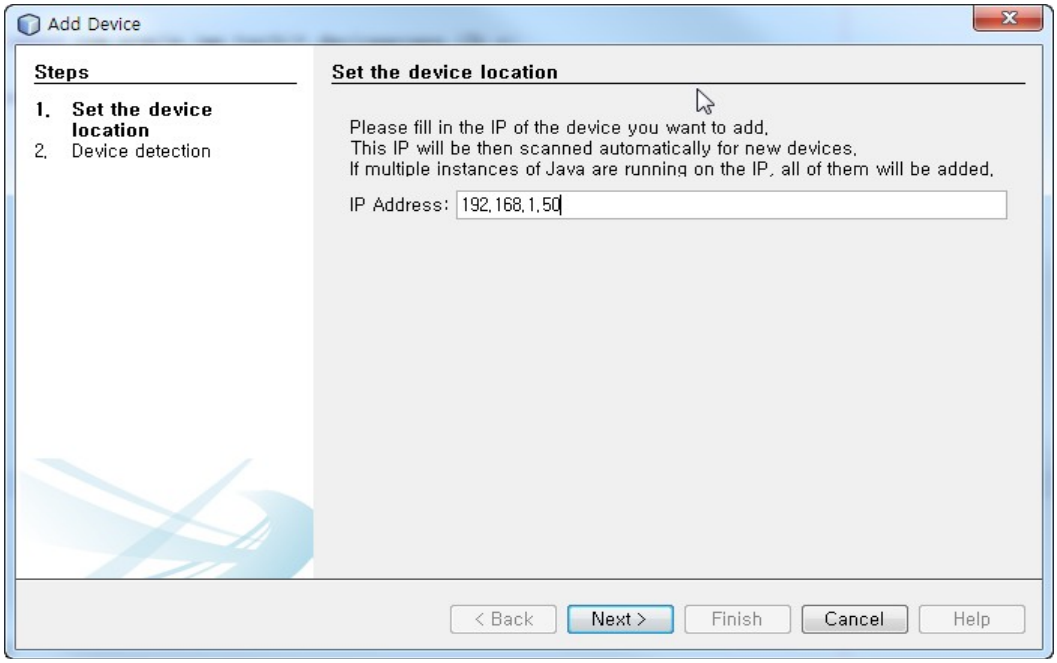
7.1. Adding the Development Board to the Device Manager

Here are the steps to add the board to the Device Manager in the Oracle Java ME SDK:

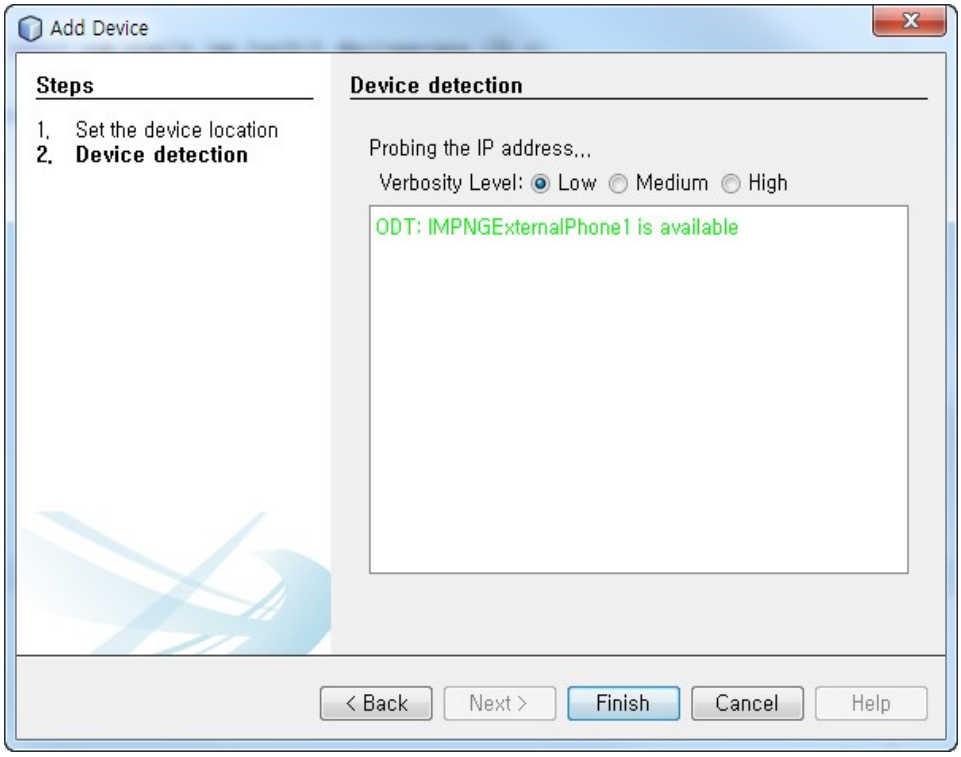
- Open TCP port 2808 in the firewall settings of your computer. The exact procedure to open a port differs depending on your version of Windows or the firewall software that is installed on your computer.
- On the NetBeans IDE, go to Tools -> Java ME -> Device Selector
  (If you don’t see “Java ME” menu, you should install Java ME SDK plugin for NetBeans)

- On the Device Selector, click on Add a Device.

- Enter the IP address of the ORION board in the IP Address field and click Next. IP address of the board can be obtained by running ‘net info’ command from the CLI.



- Click Finish on the Device Detection screen.



- The list of devices in the Device Selector should now include IMPNGExternalPhone1.
- Please refer to the following link for additional guides regarding NetBeans:
  - http://docs.oracle.com/javame/dev-tools/jme-sdk-3.2/dev-guide/deviceselector.htm#CFAJIAGC

7.2. Assigning the Development Board to Your Project

It is easy to assign the development board to your project. If you already have an existing NetBeans project with an IMlet that you wish to run or debug on the board, follow these steps:

- Right-click on your project and choose Properties.
- Select the Platform category on the properties window.
- Select IMPNGExternalDevice from the device list.

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

- Select File -> New Project.
- Select the ‘Java ME’ category and ‘Embedded Application’ in Projects.
- Provide a project name and click Next.
- Select ‘CLDC Oracle Java Platform ME 3.2’ for Emulator Platform.
- Select ‘IMPNGExternalDevice’ from the device list and click Finish.

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

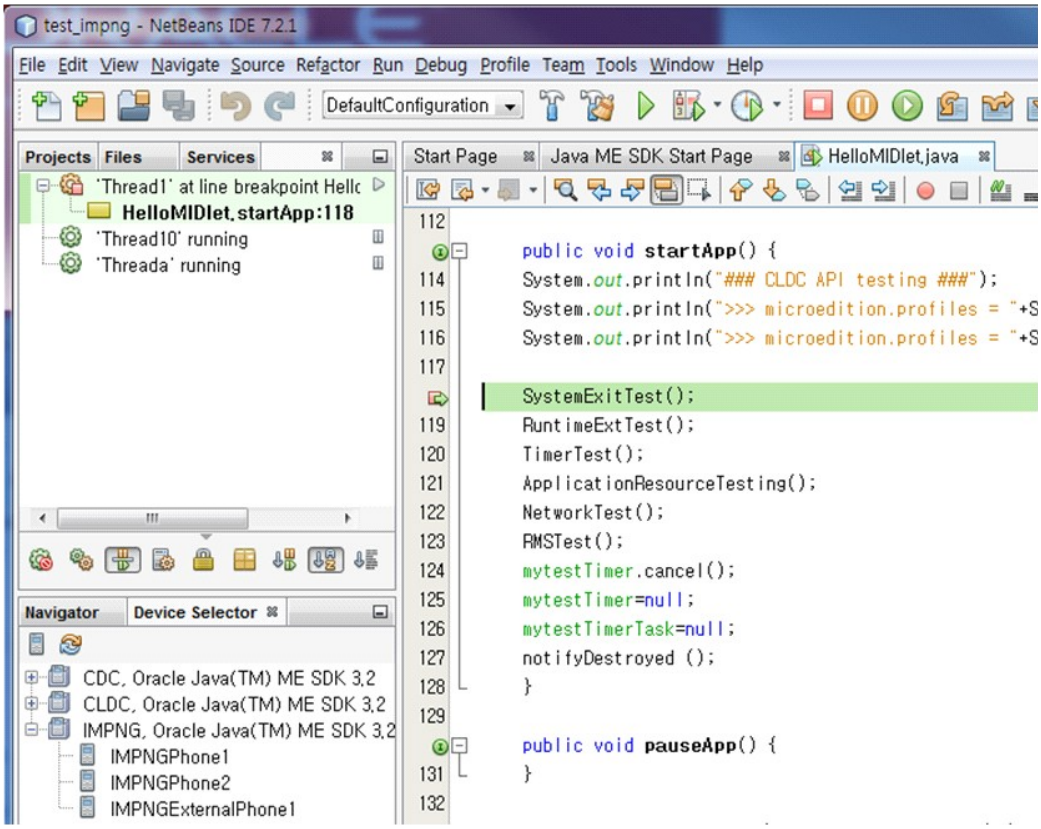
7.3. Debugging Applications on the Development Board from NetBeans

First of all, check your NetBeans' version, 7.2.1.

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

- Open your IMlet class on the NetBeans editor.
- Click once 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.
- Select Debug -> Debug Project or use the Debug button on the toolbar.

The debugger connects to the debug agent on the board and the program execution stops at your breakpoint, as shown in screenshot below.



Unless the application is designed to self-terminate, you need to manually stop the application to close the debug/run session. This can be done by pressing the stop button or by clicking the X icon in the bottom status bar of the NetBeans IDE right of the currently running application name.

7.4. Debugging Applications on the Development Board from Eclipse

Eclipse follows the similar procedure as NetBeans. Please refer to the following document for more information.

- http://docs.oracle.com/javame/dev-tools/jme-sdk-3.2/ecl/html/deviceselector.htm#CFAJIAGC

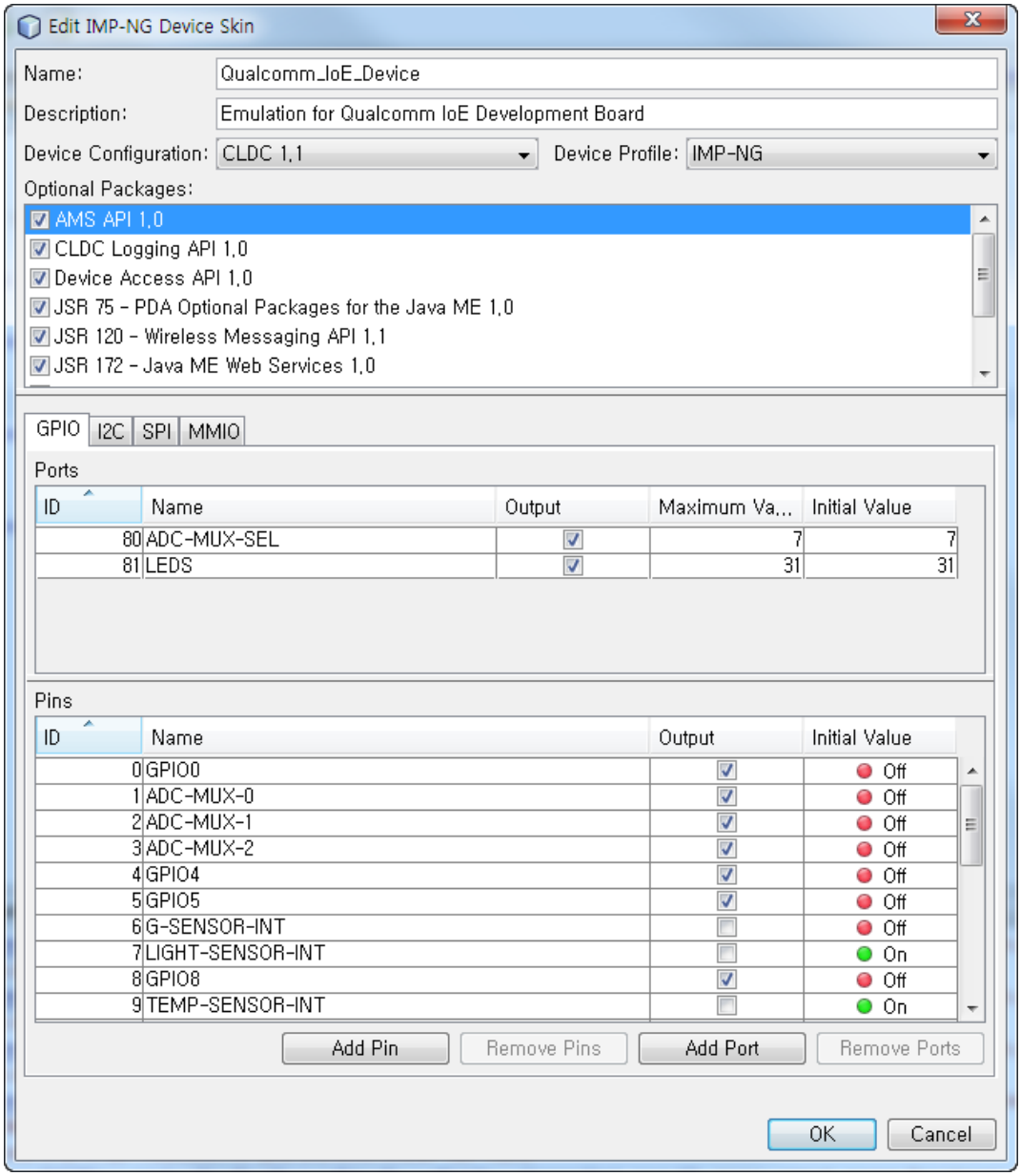
8.Using the Custom Emulation Skin

Using the Java ME SDK 3.2 feature for providing custom emulation skin, we have implemented emulation for the following on-board peripherals.

- I2C Temperature Sensor
  - Supports both default and extended mode for temperature reading
  - Supports setting of Low and High Thresholds
  - Interrupt alert when threshold is tripped
    - Manually operate external events generator
- I2C Light Sensor
  - Support Ambient Light Sensing (Continuous and once)
  - Supports setting Low and High Threshold
  - Interrupt alert when threshold is tripped
    - Manually operate external events generator
- I2C G-Sensor (Accelerometer)
- SPI G-Sensor (Accelerometer)

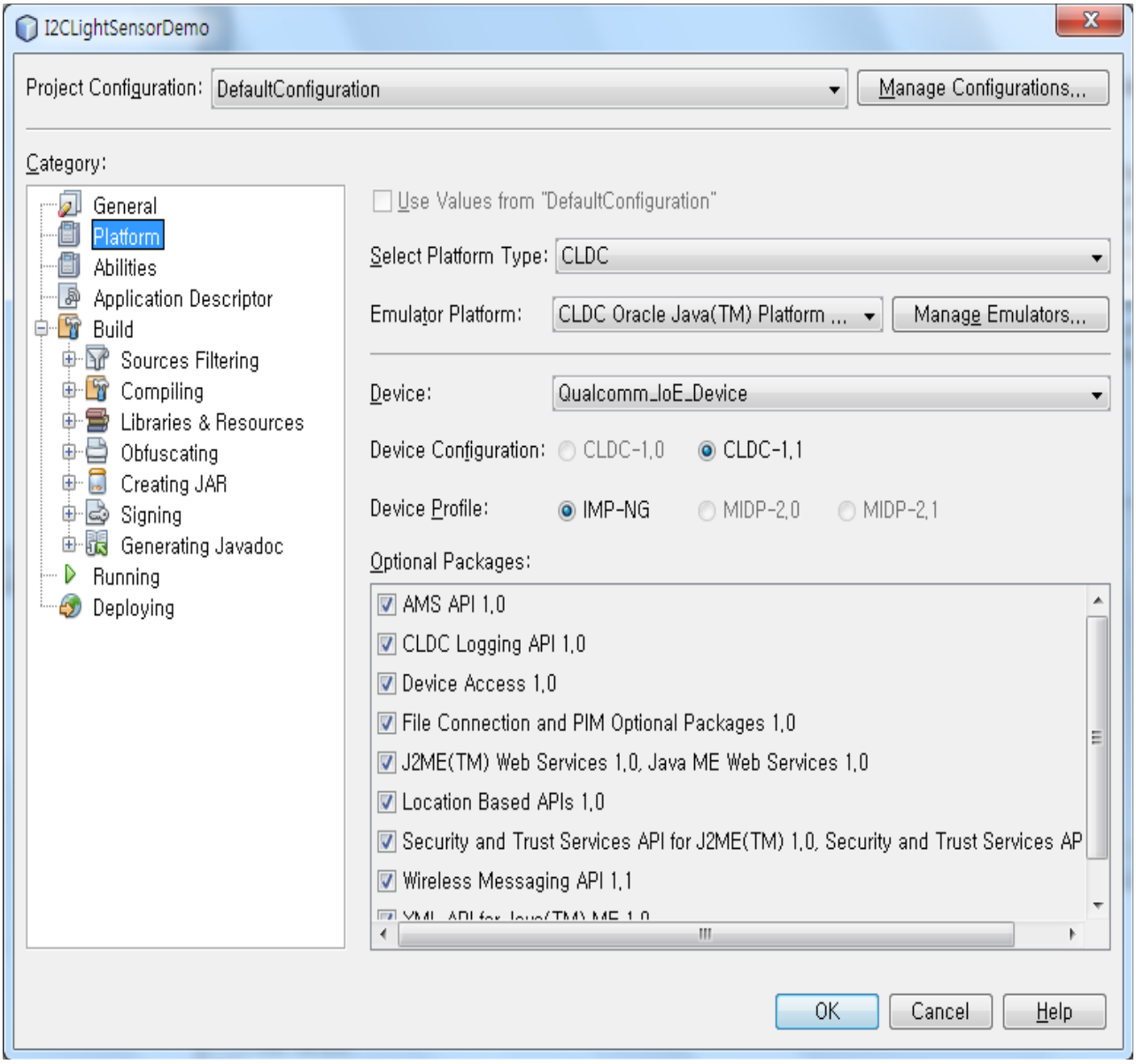
8.1. Importing the Custom Skin

Custom Skin can be imported from the NetBeans menu 'Tools > Java ME > Custom Device Skin Creator. On the 'Custom Device Skin Creator' dialog select 'Import' and open 'Qualcomm_IoE_Device.zip' file which is provided with user guide. After a successful import the skin should show up under 'IMP-NG' folder. If you select the skin and do a 'Edit' from the dialog, you should see the following diagram.

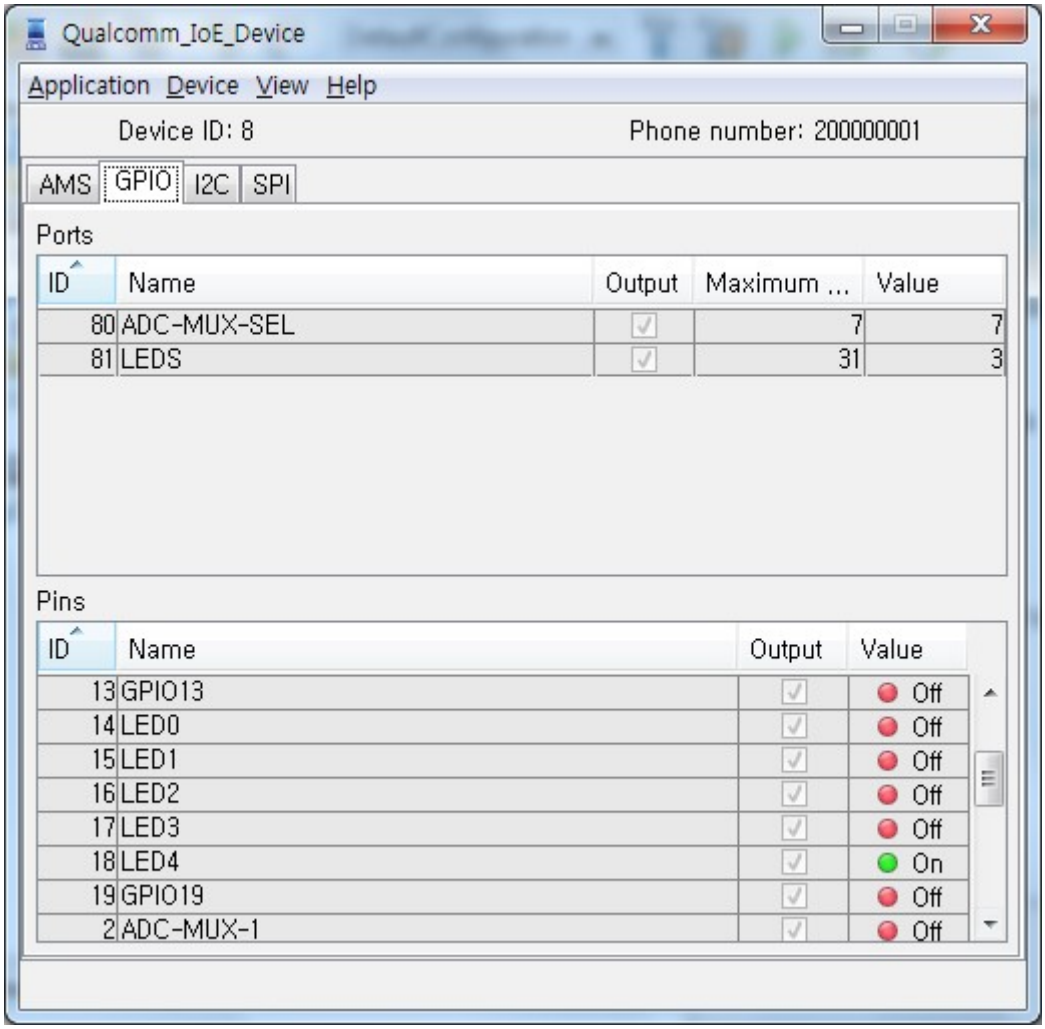


8.2. Running and Debugging Using Custom Skin

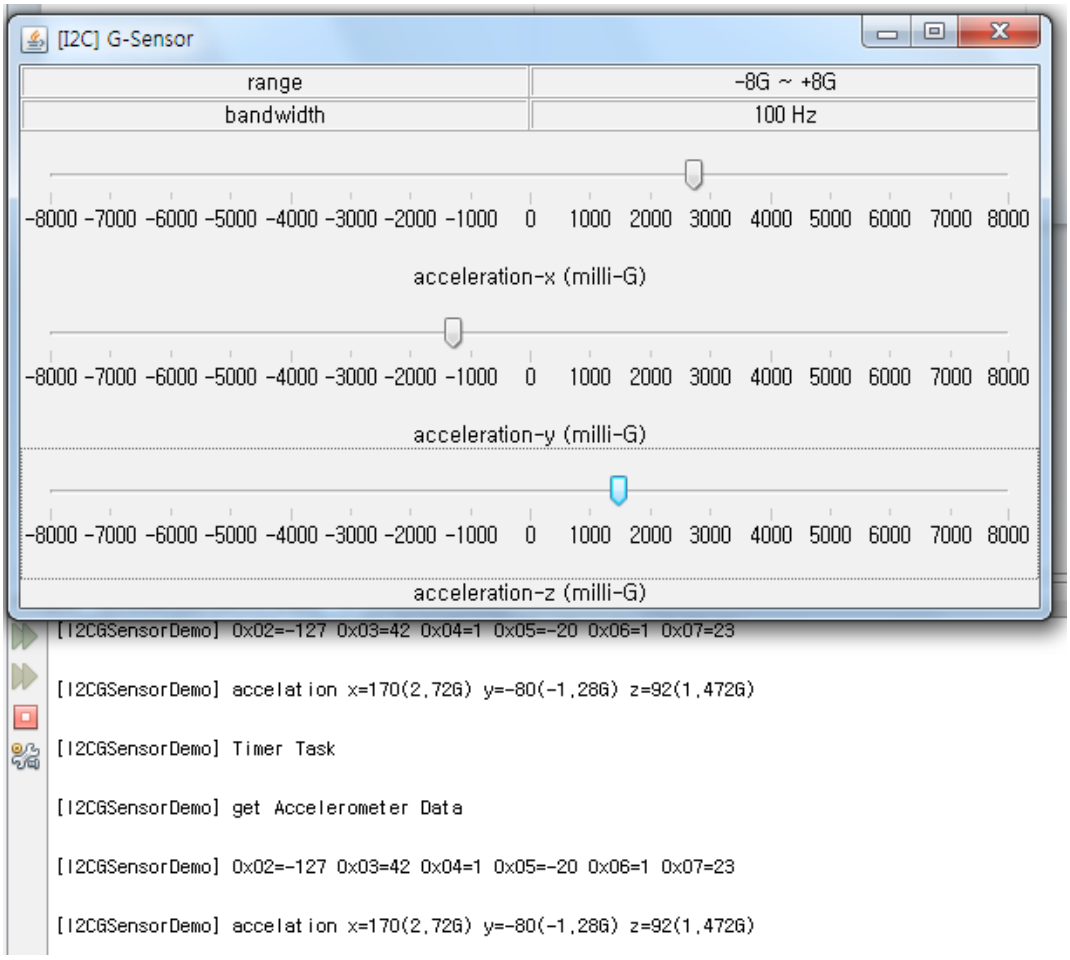
To run or debug using the custom skin, set device to “Qualcomm_IoE_Device” in “Platform” tab. And then, start running or debugging. Setting breakpoints are also available. When an IMlet using DAAPIs is launched, custom emulator launches each emulator and helps to change device values.



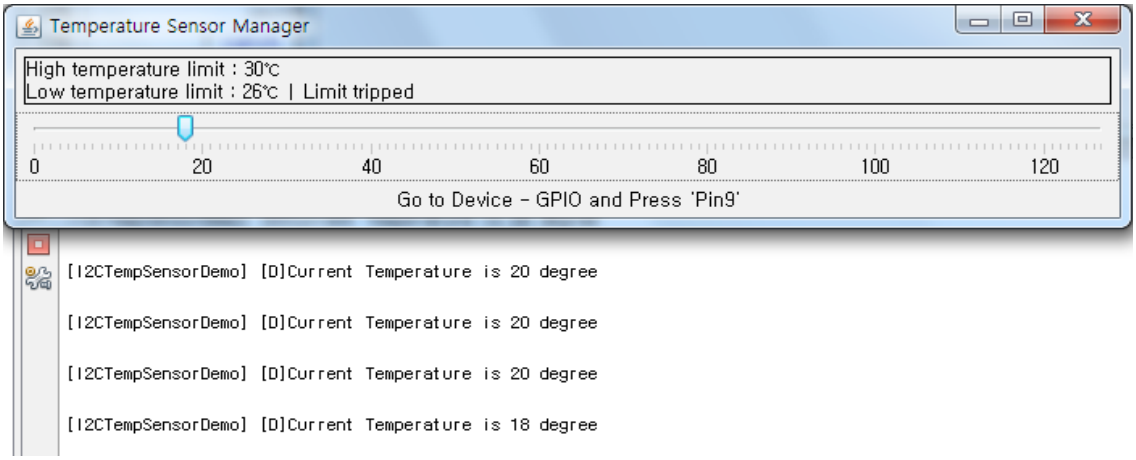
- GPIO Pin
  - After running IMlet using GPIO Pin, click “GPIO” tab
  - LED0 to LED4 are changed to ON / OFF with green / red color
  - Use LEDS or ADC-MUX-SEL for GPIO port test.



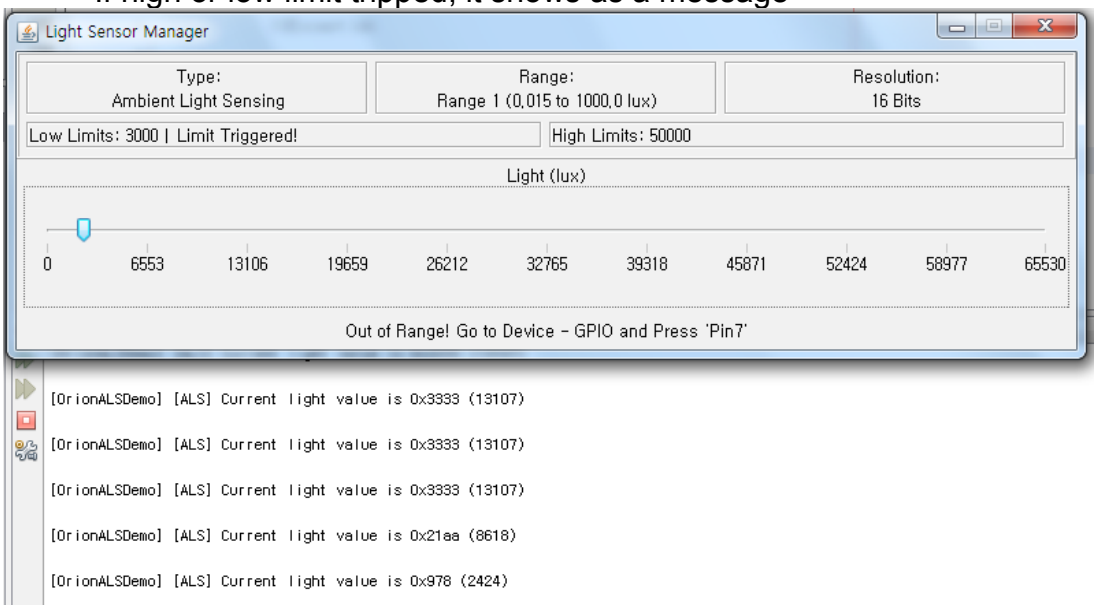
- Accelerometer Sensor
  - After running IMlet using G Sensor, G Sensor emulator is started
  - Move slider of X, Y, or Z coorinator to change x,y,z values.
  - Imlet can reads accelerometer values from the emulator through I2C or SPI.
  - For interrupt test, open Device → GPIO, click Pin 6



- Temperature Sensor
  - After running IMlet using Temperature Sensor, Temperature Sensor emulator is started
  - Move slider of temperature bar
  - IMlet reads temperature values from the emulator
  - For interrupt test, open Device → GPIO, click Pin 9
    - If high or low limit tripped, it shows as a message



- Light Sensor
  - After running IMlet using Light Sensor, Light Sensor emulator is started
  - Move slider of light value bar
  - IMlet reads light values from the emulator
  - For interrupt, open Device → GPIO, click Pin 7
    - If high or low limit tripped, it shows as a message



9. Sample Project Overview

Sample	Description	Instructions
GPIOLEDDemo	An example IMlet for LED connected via GPIO	
I2CTempSensorDemo	An example IMlet for Temperature Sensor connected via I2C bus	Mode can change through JAD-Property value - Temperature-Range : Default/Extended
I2CLightSensorDemo	An example IMlet for a light sensor via I2C interface	In JAD-Property, - Operation-Mode: ALS / IR / Proximity (Continuous) ALS / IR / Proximity (Once) - Resolution: 16, 12, 8, or 4 bits (Default is 16 bits) - Range: the full scale range, default is Range 1
I2CGSensorDemo	An example IMlet for G-Sensor(Accelerometer) connected via I2C bus	
SPIGSensorDemo	An example IMlet for G-Sensor(Accelerometer) connected via SPI bus	
ADCDemo (Device Only)	An example IMlet for ADC (Analog to Digital Converter)	
ATCmdDemo (Device Only)	An example IMlet for seding AT Commands and handling responses	
WatchdogDemo (Device Only)	An example IMlet for Watchdog	For repeatable test, add JAD Property, Oracle-MIDlet-Autostart: 1

- *Please refer datasheet document for detail informations of registers*
- *For more informations, refer to jwc_properties.reference.ini*

10. Cautions on Default Security Policy

- For easier testing and development on the development board, the device ships with all permissions allowed even for untrusted applications.
- For deploying on a commercial device, you will need to sign you applications with the proper singing certificate distributed by the manufacturer or the operator.
- Depending on the application, you may also need to handle SecurityException in cases where you don’t have permission to use privileged APIs such as device access APIs. You may need to fallback gracefully in such case.

11.Known issues

- Connecting to wireless AP using WEP authentication is not supported. Use WPA/WPA2-PSK which offer better protection.
- Receiving SMS without port is not allowed in JSR 120. Special port was designated for receiving SMS without port information. Listen for SMS with URI “sms://:65533”.
- AT Command APIs are not fully supported. Refer for more details.
- ADC,AT Command and Watchdog API need modification to run on the 3.3 binary. Refer Device Access APIs for more details.

Appendix A. Java Application Auto-start

The Oracle Java ME Embedded platform can be configured to allow Java applications to start automatically on launch of the Java runtime. The following proprietary properties should be defined in the application JAD file to enable the auto-start feature.

Note that the Oracle-MIDlet-Autostart is mandatory for autostart. The remaining properties are optional.

- Proprietary JAD properties are introduced
  - **Oracle-MIDlet-Autostart**: [0-5]
  - Oracle-MIDlet-Restart: [true|false]
  - Oracle-MIDlet-Restart-Count: [number]
- Oracle-Midlet-Autostart
  - If there is no property, or it is set to level 0, this means "do not autostart." A level between 1 and 5 determines the priority of autostart. For different IMlets with the same level, the starting order is up to implementation, such as suite name in descending order.
- Oracle-MIDlet-Restart
  - This property specifies if IMlet should be restarted on a critical failure while the JVM is running
- Oracle-MIDlet-Restart-Count
  - This property determines how many times the IMlet will be restarted. If the property is not specified, default value is 1, which means that on subsequent failures, it will not be restarted.

Appendix B. On-Board Peripherals Access Information for Java Application

This appendix describes the proper ID and names for the various peripheral ports and buses for the Qualcomm IoE Development board, which are accessible using the Device
Access APIs.

Note that any IMlet that accesses the Device Access APIs must be digitally signed using a trusted certificate authority. The default security policy for Qualcomm IoE Development board was changed to allow even unsigned applications to run without throwing a SecurityException. When commercializing the platform, this SHUOULD NOT be the default behavior and proper security and permission configuration should be done referencing relavant Java ME product documents.

* Also please refer to the 'jwc_properties.reference.ini' file for the default configuration properties for the peripherals as well.

GPIO Pins

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
0	GPIO0	26	Output, initial value = 0
1	ADC-MUX-0	25	Output, initial value = 1
2	ADC-MUX-1	31	Output, initial value = 1
3	ADC-MUX-2	17	Output, initial value = 1
4	GPIO4	32	Output, initial value = 0
5	GPIO5	28	Output, initial value = 0
6	G-SENSOR-INT	27	Input, active high,edge detection
7	LIGHT-SENSOR-INT	30	Input, active low, level detection
8	GPIO8	38	Output, initial value = 0

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
9	TEMP-SENSOR-INT	33	Input, active low, edge detection
10	RELAY1	18	Output, initial value = 0
11	RELAY2	24	Output, initial value = 0
12	GPIO12	29	Output, initial value = 0
13	GPIO13	35	Output, initial value = 0
14	LED1	13	Output, initial value = 0
15	LED0	34	Output, initial value = 0
16	LED3	12	Output, initial value = 0
17	LED2	16	Output, initial value = 0
18	LED4	36	Output, initial value = 0
19	GPIO19	15	Output, initial value = 0
20	GPIO20	10	Output, initial value = 0
21	GPIO21	14	Output, initial value = 0
22	GPIO22	11	Output, initial value = 0
23	GPIO23	9	Output, initial value = 0
24	GPIO24	37	Output, initial value = 0

GPIO Ports

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
80	ADC-MUX-SEL	25,31,17	Output,MaxVal: 7
81	LEDS	34,13,16,12,36	Output,MaxVal: 31

ADC

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
100	VTHERM_N	VTHERM_N	sampling interval of channel in msec = 500 maximum sampling interval of channel in msec = 5000 minimum sampling interval of channel in msec = 100 maximum value of this channel = 4095 minimum value of this channel = 0 Reference Voltage of this ADC channel(ex, 2.2V = 2200) = 2200
101	HKAIN1	HKAIN1	sampling interval of channel in msec = 500 maximum sampling interval of channel in msec = 5000 minimum sampling interval of channel in msec = 100 maximum value of this channel = 4095 minumum value of this channel = 0 Reference Voltage of this ADC channel = 2200

I^2C

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
300	G-SENSOR	G-Sensor	Address: 56
301	LIGHT-SENSOR	Light-Sensor	Address: 68
302	TEMP-SENSOR	Temp-Sensor	Address: 75
303	BATTERY-GAUGE	Battery-Gauge	Address: 85

SPI

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
300	G-SENSOR	G sensor	Slave address: 56

Watchdog

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
500	WDG	Internal Watchdog	

Appendix C. Error Messages shown during Network Setup

Error Message	Description
`WLAN ERROR: Wireless AP Not found! SSID = [SSID]`	wireless AP is not found . Check your SSID properties, “system.wlan.ap.ssid” in jwc_properties.ini
`WLAN ERROR: Connection attempt failed with code = [error code]!`	Connection attempt failed. Check the error code.
`WLAN ERROR: Failed read password from settings!`	The password of wireless AP is not found. Check your SSID properties, “system.wlan.ap.passwd” in jwc_properties.ini.
`WLAN ERROR: Unsupported Authentication!`	The auth type of wireless AP is wrong. Check you’re an auth type and encryption.
`WLAN ERROR: NOT CONNECTED - Authentication failure!`	The auth info is wrong. Check your auth information.
`WLAN ERROR: NOT CONNECTED - Wireless AP unreachable!`	There is no wireless access point in your coverage. Check your wireless information.
`WLAN ERROR: NOT CONNECTED - WLAN security failure!`	Security issue. Check your security setting of wireless AP.
`WLAN ERROR: NOT CONNECTED - WLAN general failure!`	General failure.
`WLAN ERROR: NOT CONNECTED - Failed with reason = [error code]!`	Unknown error. Check the error code.

Appendix D. AMS Installer Error Codes

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

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

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

Constant	Error Code	Description
INVALID_NATIVE_LIBRARY	85	A native library contained within the JAR cannot be loaded.
INVALID_PACKAGING	87	A dependency cannot be satisfied.
INVALID_PAYMENT_INFO	58	Indicates that the payment information provided with the IMlet suite is incomplete or incorrect.
INVALID_PROVIDER_CERT	7	The signature of the content provider certificate is invalid.
INVALID_RMS_DATA_TYPE	76	The server did not have a resource with the correct type or the JAD downloaded has the wrong media type.
INVALID_RMS_DATA_URL	73	The RMS data file URL is invalid.
INVALID_SERVICE_EXPORT	86	A LIBlet that exports a service with a LIBlet Services attribute does not contain the matching service provider configuration information.
INVALID_SIGNATURE	9	The signature of the JAR is invalid.
INVALID_VALUE	29	A value for an attribute is not formatted correctly.
INVALID_VERSION	16	The format of the version is invalid.
IO_ERROR	102	A low-level hardware error has occurred.
JAD_MOVED	34	The JAD URL for an installed suite is different than the original JAD URL.
JAD_NOT_FOUND	2	The JAD was not found.
JAD_SERVER_NOT_FOUND	1	The server for the JAD was not found.
JAR_CLASSES_VERIFICATION_FAILED	56	Not all classes within JAR package can be successfully verified with class verifier.
JAR_IS_LOCKED	100	Component or MIDlet or IMlet suite is locked by the system.
JAR_NOT_FOUND	20	The JAR was not found at the URL given in the JAD.
JAR_SERVER_NOT_FOUND	19	The server for the JAR was not found at the URL given in the JAD.
JAR_SIZE_MISMATCH	31	The JAR downloaded was not the same size as given in the JAD.
MISSING_CONFIGURATION	41	The configuration is missing from the manifest.

Constant	Error Code	Description
MISSING_DEPENDENCY_HASH	67	A dependency hash code is missing.
MISSING_DEPENDENCY_JAD_URL	66	A dependency JAD URL is missing.
MISSING_JAR_SIZE	21	The JAR size is missing.
MISSING_JAR_URL	18	The URL for the JAR is missing.
MISSING_PROFILE	42	The profile is missing from the manifest.
MISSING_PROVIDER_CERT	4	The content provider certificate is missing.
MISSING_SUITE_NAME	13	The name of MIDlet or IMlet suite is missing.
MISSING_VENDOR	14	The vendor is missing.
MISSING_VERSION	15	The version is missing.
NEW_VERSION	32	This suite is newer that the one currently installed.
NO_ERROR	0	No error.
NOT_YET_VALID_PROVIDER_CERT	89	A certificate is not yet valid.
NOT_YET_VALID_CA_KEY	90	A CA's public key is not yet valid.
OLD_VERSION	17	This suite is older that the one currently installed.
OTHER_ERROR	103	Other errors.
PROXY_AUTH	51	Indicates that the user must first authenticate with the proxy.
PUSH_CLASS_FAILURE	48	The class in a push attribute is not in MIDlet-<n> attribute.
PUSH_DUP_FAILURE	45	The connection in a push entry is already taken.
PUSH_FORMAT_FAILURE	46	The format of a push attribute has an invalid format.
PUSH_PROTO_FAILURE	47	The connection in a push attribute is not supported.
REVOKED_CERT	62	The certificate has been revoked.
RMS_DATA_DECRYPT_PASSWORD	83	Indicates that a password is required to decrypt RMS data.
RMS_DATA_ENCRYPT_PASSWORD	84	Indicates that a password is required to encrypt RMS data.

Constant	Error Code	Description
RMS_DATA_NOT_FOUND	75	The RMS data file was not found at the specified URL.
RMS_DATA_SERVER_NOT_FOUND	74	The server for the RMS data file was not found at the specified URL.
RMS_INITIALIZATION_FAILURE	81	Failure to import RMS data.
SUITE_NAME_MISMATCH	25	The MIDlet or IMlet suite name does not match the one in the JAR manifest.
TOO_MANY_PROPS	53	Indicates that either the JAD or manifest has too many properties to fit into memory.
TRUSTED_OVERWRITE_FAILURE	52	Indicates that the user tried to overwrite a trusted suite with an untrusted suite during an update.
UNAUTHORIZED	33	Web server authentication failed or is required.
UNKNOWN_CA	6	The certificate authority (CA) that issued the content provider certificate is unknown.
UNKNOWN_CERT_STATUS	63	The certificate is unknown to OCSP server.
UNSUPPORTED_CERT	10	The content provider certificate has an unsupported version.
UNSUPPORTED_CHAR_ENCODING	61	Indicates that the character encoding specified in the MIME type is not supported.
UNSUPPORTED_PAYMENT_INFO	57	Indicates that the payment information provided with the MIDlet or IMlet suite is incompatible with the current implementation.
UNTRUSTED_PAYMENT_SUITE	59	Indicates that the MIDlet or IMlet suite has payment provisioning information but it is not trusted.
VENDOR_MISMATCH	27	The vendor does not match the one in the JAR manifest.
VERSION_MISMATCH	26	The version does not match the one in the JAR manifest.