

Oracle® Java ME Embedded

Getting Started Guide for the Reference Platform (Intel Galileo
Gen2)

Release 8.3 Developer Preview

E71182-02

July 2016

Beta Draft

ORACLE CONFIDENTIAL.

For authorized use only.

Do not distribute to third parties.

E71182-02

Copyright © 2016, Oracle and/or its affiliates. All rights reserved.

Primary Author:

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

This documentation is in preproduction status and is intended for demonstration and preliminary use only. It may not be specific to the hardware on which you are using the software. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to this documentation and will not be responsible for any loss, costs, or damages incurred due to the use of this documentation.

The information contained in this document is for informational sharing purposes only and should be considered in your capacity as a customer advisory board member or pursuant to your beta trial agreement only. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle.

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle Master Agreement, Oracle License and Services Agreement, Oracle PartnerNetwork Agreement, Oracle distribution agreement, or other license agreement which has been executed by you and Oracle and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

Contents

Preface	v
Audience	v
Related Documents.....	v
Shell Prompts.....	v
Conventions.....	v
1 Installing the Java ME Embedded Software on the Intel Galileo Gen2 Board	
Required Hardware and Software Items	1-1
Installing Yocto Linux Operating System	1-2
Connecting to the Galileo Gen2 Board	1-2
Structure of the Oracle Java ME Embedded Software	1-3
Installing the Java ME Embedded Software on the Intel Galileo Gen2 Board	1-3
Running Java ME on the Galileo Board.....	1-4
Editing the grub.conf File	1-5
Downloading and Installing the PuTTY Terminal Emulator Program	1-6
2 Installing and Running Applications on the Intel Galileo Gen2 Board	
Ways of Using the Java Runtime on the Intel Galileo Gen2 Board	2-1
Run of IMlets on Intel Galileo Gen2 Using the Command Shell	2-1
An Example of Managing Application Life Cycle with Shell Commands.....	2-2
Purpose of the Developer Agent Program on the Desktop	2-3
Configuring Client Mode Connection.....	2-3
Installing the Developer Agent Program	2-3
Starting the Developer Agent in a Server Mode	2-3
Starting the Developer Agent in a Client Mode.....	2-4
Running IMlets on Intel Galileo Gen2 Using the AMS CLI	2-4
Making a Raw Connection to the AMS CLI	2-5
Lists of Commands.....	2-6
An Example of Managing Application Life Cycle with AMS Commands.....	2-6
NetBeans and the Intel Galileo Gen2 Board	2-7
Required Software for Using the Intel Galileo Gen2 Board with NetBeans.....	2-7
Adding the Intel Galileo Gen2 Board to the Device Connection Manager.....	2-8

A Intel Galileo Gen2Device I/O Preconfigured List

Functionality of the galileo_pin_config Script.....	A-1
Reference Pin Diagram	A-2
GPIO Pins.....	A-2
SPI	A-4
I2C	A-5
UART.....	A-5
Structure of the Device I/O Configuration File	A-6

Glossary

Preface

This guide describes how to install Oracle Java ME Embedded software onto a Intel Galileo Gen2 device.

Audience

This guide is for developers who want to run Oracle Java ME Embedded software on a Intel Galileo Gen2 device.

Related Documents

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

Shell Prompts

Shell	Prompt
Windows	<i>directory></i>
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.
<i>italic</i>	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.

Installing the Java ME Embedded Software on the Intel Galileo Gen2 Board

Learn how to install the Java ME Embedded software 8.3 onto an Intel Galileo Gen2 board, configure the Java ME Embedded system, connect to the command-line and logging interfaces, and perform basic configuration tasks.

Topics

- [Required Hardware and Software Items](#)
- [Installing Yocto Linux Operating System](#)
- [Connecting to the Galileo Gen2 Board](#)
- [Structure of the Oracle Java ME Embedded Software](#)
- [Installing the Java ME Embedded Software on the Intel Galileo Gen2 Board](#)
- [Running Java ME on the Galileo Board](#)
- [Editing the grub.conf File](#)
- [Downloading and Installing the PuTTY Terminal Emulator Program](#)

Required Hardware and Software Items

This section describes the hardware and software items that are required for developing on the Intel Galileo Gen2 board.

- Intel Galileo Gen2 board
- Yocto Linux v.2.1 Operating System image
- Oracle Java ME Embedded 8.3 distribution
- Oracle Java ME SDK 8.3 (optionally)

Note:

You have two options to work with the Intel Galileo Gen2 board: with or without using the Java ME SDK.

- A 7-15-V DC power supply.
- A micro SD card of 2GB or greater up to 32 GB

- Ethernet cable with an RJ-45 connector, as well as a connection to a network with a DHCP server.
- A terminal emulator program, such as PuTTY, if you wish to connect to the board using the Application Management System (AMS) interface

Optionally, you can use a 6-pin Serial to Type A USB cable (FTDI cable # TTL-232R-3V3 is recommended) to connect to the board.

Installing Yocto Linux Operating System

You must prepare a bootable SD card for your Intel Galileo Gen2 board by downloading and installing the Yocto Linux v2.1 operating system. After that you need to install Oracle Java ME Embedded software.

1. Download the Yocto Linux v2.1 operating system image to your desktop from the following location.

<http://downloadmirror.intel.com/24738/eng/iotdk-galileo-image.bz2>

2. Use 7-Zip to unzip the distribution file, which creates a single disk image file.

On Windows, use 7-Zip utility.

On Linux, use `bunzip2` or `tar -xf <file>` to unzip the archive.

You now have a file called `iotdk-galileo-image`.

3. Mount the SD card to the desktop, and use a utility to create a bootable SD card by writing the disk image file to the SD card.

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

- For the Windows operating system, you can use the Disk Image Write utility located at <https://sourceforge.net/projects/win32diskimager/>. For more details, see [Getting Started with the Intel Galileo Board on Windows](#).
- For the Mac platform, see [Getting Started with the Intel Galileo Board on Mac OS X](#).
- For the Linux platforms, see [Getting Started with the Intel Galileo Board on Linux](#).

4. Eject the SD card from your host computer.

Connecting to the Galileo Gen2 Board

To connect to the Intel Galileo Gen2 board, follow these steps:

1. Insert a bootable SD card into your Intel Galileo Gen2 board.
2. Connect the RJ-45 network cable to the board.
3. Connect power to the board.

The green light ON on the board should glow, then the USB light should also glow and in a few seconds, the green light SD should start blinking. The blinking green light indicates that the board is booting Linux.

4. If the Linux installation was successful, the Intel Galileo Gen2 board obtains an IP address.
5. Scan the network to find out the Intel Galileo Gen2 IP address.

For example, run the `nmap` command.

```
nmap -sn <IP Address>/<Subnet Mask | grep Intel -B2
```

You need to know the IP Address of your board to access and control the board remotely. Remember this IP address for further use.

6. Start an LXTerminal program and login to the board.

Linux users can use the `ssh` command to login to the board. Note that there is no `sudo` command in Intel Galileo Gen2 board.

The user name is `root`, a password is empty, i.e no password is required.

Structure of the Oracle Java ME Embedded Software

The Oracle Java ME Embedded software for the Intel Galileo Gen2 board is distributed as a ZIP archive.

The user has an option to run a Developer Agent program on the desktop under Windows or Linux. Commands that are sent to the board from the host desktop are no longer sent directly across the network. Instead, they are sent to the Developer Agent program, which transmits all communication to and from the Oracle Java ME Embedded executable file on the Intel Galileo Gen2 board.

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

- `/appdb`: This directory is used on the Intel Galileo Gen2 and contains internal Java libraries.
- `/bin`: This directory is used on the Intel Galileo Gen2 and contains executables and the `jwc_properties.ini` file.
- `/legal`: This directory contains important legal documentation.
- `/lib`: This directory contains the files needed to compile IMlets on the Intel Galileo Gen2 board.
- `/toolkit-lib`: This directory contains Java heap dumps and Java Heap Analyzer Tool (HAT).
- `/util`: This directory contains the Developer Agent program.

Installing the Java ME Embedded Software on the Intel Galileo Gen2 Board

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

1. Use an `sftp` client or `scp` command to transfer a copy of the Oracle Java ME Embedded archive to the Intel Galileo Gen2 board.

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

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

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

<http://www.putty.org/>

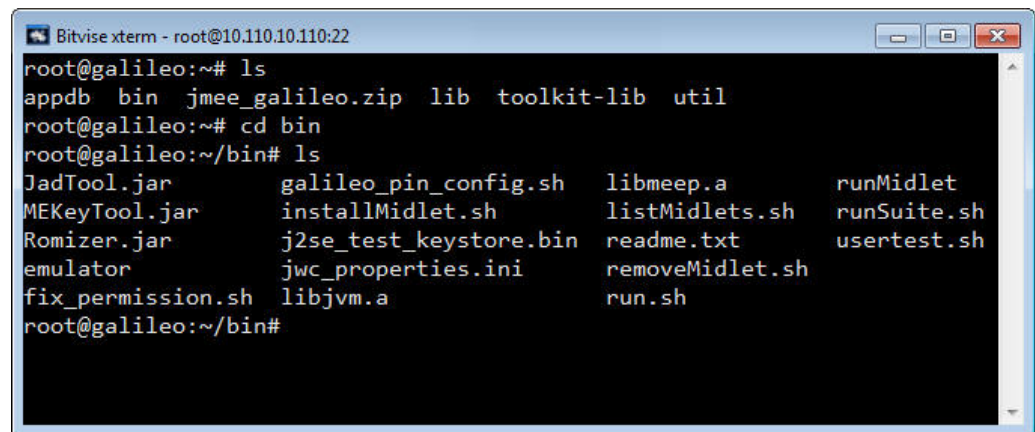
2. After the ZIP archive is transferred, either go directly to the keyboard and the mouse connected to the Intel Galileo Gen2 board, or start a secure shell script on your desktop to connect to the board using the following command:

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

3. Unzip the archive on the Intel Galileo Gen2 board.
4. Change to the `bin` directory.

The contents of the `bin` directory are shown in [Figure 1-1](#).

Figure 1-1 Intel Galileo Gen2 Board bin Directory



```
Bitvise xterm - root@10.110.10.110:22
root@galileo:~# ls
appdb bin jmee_galileo.zip lib toolkit-lib util
root@galileo:~# cd bin
root@galileo:~/bin# ls
JadTool.jar          galileo_pin_config.sh  libmeep.a           runMidlet
MEKeyTool.jar       installMidlet.sh      listMidlets.sh     runSuite.sh
Romizer.jar         j2se_test_keystore.bin  readme.txt         usertest.sh
emulator            jwc_properties.ini    removeMidlet.sh
fix_permission.sh   libjvm.a              run.sh
```

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

```
root@galileo:~/bin# fix_permission.sh
```

Running Java ME on the Galileo Board

Ensure that you installed the bootable SD card in your Intel Galileo Gen2 board, connect the board to the power source and to Ethernet.

1. Run an SSH shell and login to the board using its IP Address.

Note that the default user name is `root` without a password.

2. CD to the `bin` directory.
3. Run the `galileo_pin_config.sh` script.

```
root@galileo:~/bin# ./galileo_pin_config.sh
```

You need to run the `galileo_pin_config.sh` script once after a reboot. For more information, see [Functionality of the galileo_pin_config Script](#).

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

```
root@galileo:~/bin# ./fix_permission.sh
```

The `fix_permission.sh` provides the following functionality:

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

5. Run Java ME on the Intel Galileo Gen2 board.

```
root@galileo:~/bin# ./usertest.sh
```

You will see the following output.

```
Java is starting. Press Ctrl-C to exit
```

Editing the grub.conf File

The `grub.conf` file modification is required for connecting SPI devices to the Intel Galileo Gen2 board. You must add a special kernel parameter to enable the Linux SPI driver to control the Chip Select (CS) pin for SPI input/output.

To edit the `grub.conf` file, perform the following steps.

1. Login to the Intel Galileo Gen2 board.

2. Run the `cat` command to check the current setting.

```
root@galileo:~# cat /sys/module/intel_qrk_plat_galileo_gen2/
parameters/gpio_cs
```

The respond is 0.

3. CD to the root directory on the SD card.

```
root@galileo:~# cd ./
```

4. Open the `/media/mmcblk0p1/boot/grub/grub.conf` file for editing.

5. Append the kernel parameter `intel_qrk_plat_galileo_gen2 gpio_cs=1` to the end of the line beginning with `kernel`

```
kernel /bzImage root=/dev/mmcblk0p2 rootwait
console=ttyS1,115200n8 ... intel_qrk_plat_galileo_gen2 gpio_cs=1
```

6. Reboot the board.

7. Run the `cat` command to check the setting again.

```
root@galileo:~# cat /sys/module/intel_qrk_plat_galileo_gen2/
parameters/gpio_cs
```

The respond is 1 which means that you set the parameter correctly.

Downloading and Installing the PuTTY Terminal Emulator Program

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

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

<http://www.putty.org/>

Note:

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

Installing and Running Applications on the Intel Galileo Gen2 Board

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

Topics

- [Ways of Using the Java Runtime on the Intel Galileo Gen2 Board](#)
- [Run of IMlets on Intel Galileo Gen2 Using the Command Shell](#)
- [An Example of Managing Application Life Cycle with Shell Commands](#)
- [Purpose of the Developer Agent Program on the Desktop](#)
- [Installing the Developer Agent Program](#)
- [Starting the Developer Agent in a Server Mode](#)
- [Starting the Developer Agent in a Client Mode](#)
- [Running IMlets on Intel Galileo Gen2 Using the AMS CLI](#)
- [NetBeans and the Intel Galileo Gen2 Board](#)

Ways of Using the Java Runtime on the Intel Galileo Gen2 Board

There are several ways to use the Oracle Java ME Embedded platform on the Intel Galileo Gen2 board.

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

Run of IMlets on Intel Galileo Gen2 Using the Command Shell

You can run IMlets directly on the Intel Galileo Gen2 board by using the commands shown in [Table 2-1](#).

Table 2-1 Intel Galileo Gen2Shell Commands

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

Note:

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

An Example of Managing Application Life Cycle with Shell Commands

The following is a typical example of using the commands to install, list, run, and remove an Oracle Java ME Embedded application on the Intel Galileo Gen2 Pi board. Most commands can be terminated with the Ctrl-C key combination if they become unresponsive.

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

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

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

```
root@galileo:~/bin# $ ./listMidlets.sh
Java is starting. Press Ctrl-C to exit
detect_fb_type: unknown device type
Suite: 2
  Name: EmbeddedTestProject
  Version: 1.0
  Vendor: Vendor
  MIDlets:
    MIDlet: GPIODemo
```

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

```
root@galileo:~/bin# ./runSuite.sh 2
Java is starting. Press Ctrl-C to exit
Starting - GPIODemo
```

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

```
root@galileo:~/bin# ./removeMidlet.sh 2
Java is starting. Press Ctrl-C to exit
Suite removed
root@galileo:~/bin#
```

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

```
root@galileo:~/bin# ./listMidlets.sh
Java is starting. Press Ctrl-C to exit
No suites installed
```

Purpose of the Developer Agent Program on the Desktop

The Developer Agent program runs on the Windows or Linux desktop and transmits all communication to and from the Oracle Java ME Embedded executable file on the Intel Galileo Gen2 board.

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

Configuring Client Mode Connection

To configure a client mode connection, edit the `jwc_properties.ini` file in the `bin` directory on the Intel Galileo Gen2 board as follows.

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

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

Installing the Developer Agent Program

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

To install the Developer Agent program, follow these steps:

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

Starting the Developer Agent in a Server Mode

When using the Developer Agent in a server mode, it is important that you first started the Java runtime on the Intel Galileo Gen2 board to allow access to the AMS.

To start the Developer Agent program, follow these steps:

1. Change to the `bin` directory on the Intel Galileo Gen2 board and run the `./fix_permission.sh` command:

```
root@galileo:~/bin# ./fix_permission.sh
```

2. Run the `./usertest.sh` command

```
root@galileo:~/bin# ./usertest.sh
```

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

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

You should see an output similar to the following:

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

After the Developer Agent starts, use the AMS CLI.

Starting the Developer Agent in a Client Mode

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

To start the Developer Agent program, use these steps:

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

```
root@galileo:~/bin# ./usertest.sh
```

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

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

You should see an output similar to the following:

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

After the Developer Agent starts, use the AMS CLI.

Running IMlets on Intel Galileo Gen2 Using the AMS CLI

To run IMlets on the Intel Galileo Gen2 board using the AMS CLI, you must first make a raw connection to the AMS CLI and then use commands.

This section contains the following topics:

- [Making a Raw Connection to the AMS CLI](#)
- [Lists of Commands](#)
- [An Example of Managing Application Life Cycle with AMS Commands](#)

Making a Raw Connection to the AMS CLI

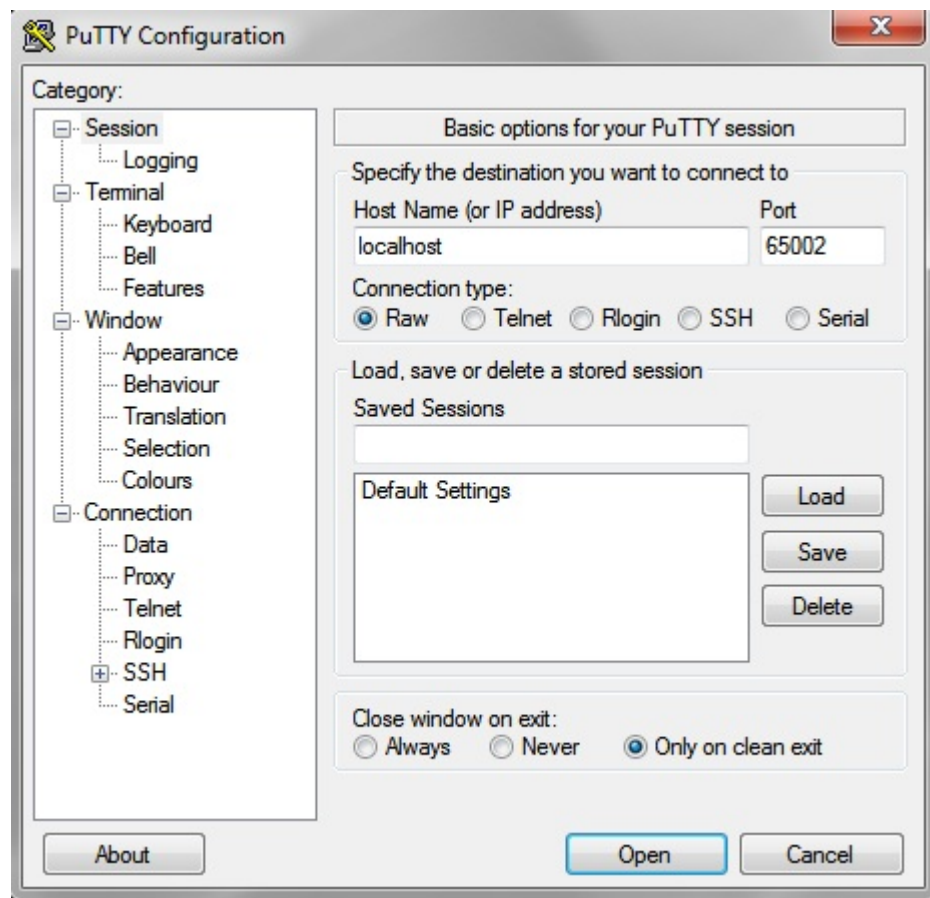
Before making a raw connection to the AMS CLI ensure that you started the Developer Agent program on the desktop host and the Java runtime - on the Intel Galileo Gen2 board unless the Developer Agent was started automatically by Java ME SDK.

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

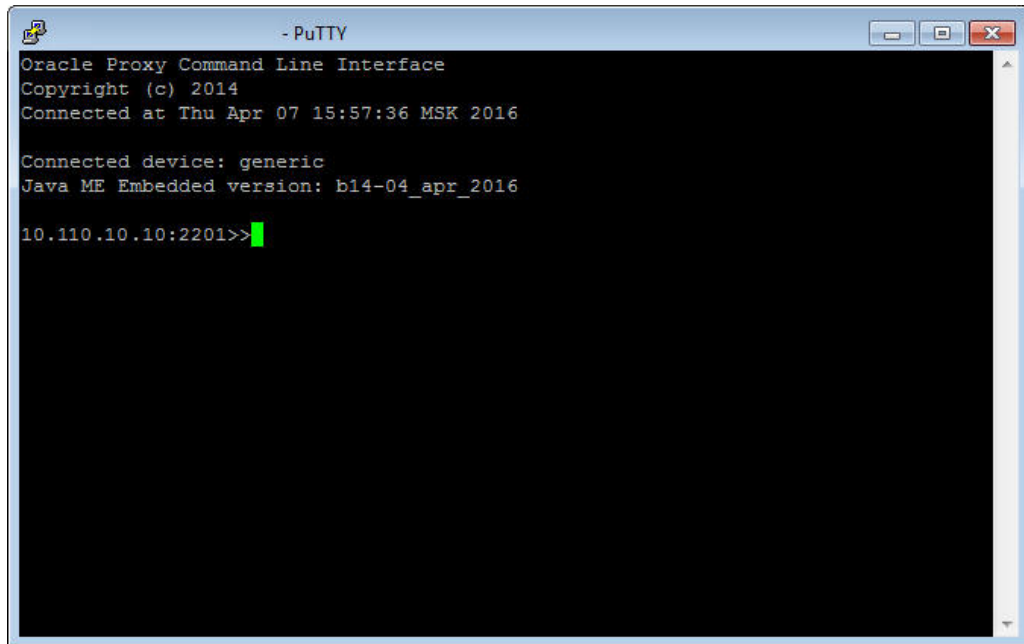
1. Start a PuTTY executable file on your desktop computer.
2. Create raw socket connections to the IP address of the host running the Developer Agent, and port 65002.

A connection to localhost and the port 65002 is shown in [Figure 2-1](#).

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



The window from port 65002 provides a CLI as shown in [Figure 2-1](#).

Figure 2-2 Command-Line Interface to Intel Galileo Gen2

```
- PuTTY
Oracle Proxy Command Line Interface
Copyright (c) 2014
Connected at Thu Apr 07 15:57:36 MSK 2016

Connected device: generic
Java ME Embedded version: b14-04_apr_2016

10.110.10.10:2201>>
```

Note:

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

Lists of Commands

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

An Example of Managing Application Life Cycle with AMS Commands

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

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

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

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

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

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

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

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

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

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

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

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

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

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

NetBeans and the Intel Galileo Gen2 Board

Topics:

- [Required Software for Using the Intel Galileo Gen2 Board with NetBeans](#)
- [Adding the Intel Galileo Gen2 Board to the Device Connection Manager](#)

Required Software for Using the Intel Galileo Gen2 Board with NetBeans

Running and debugging IMlet projects on the Intel Galileo Gen2 board using the NetBeans IDE 8.1 requires the following software:

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

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

Note:

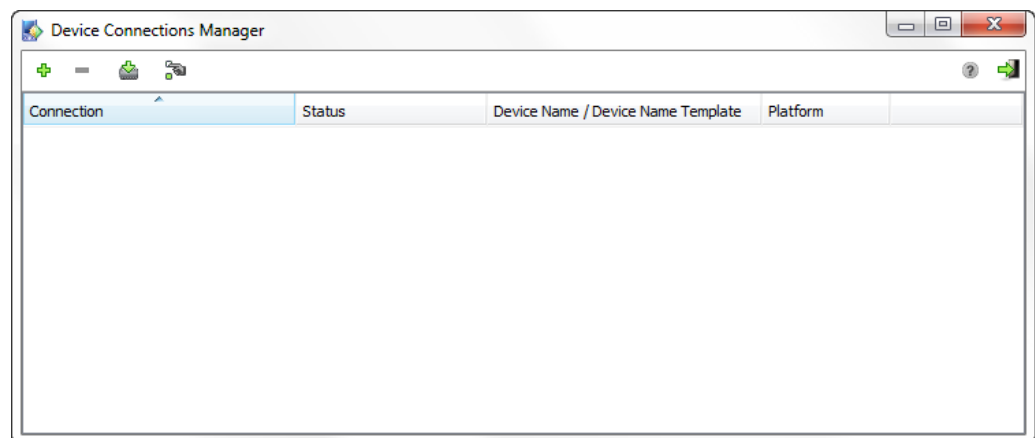
This chapter assumes that the Intel Galileo Gen2 board is already set up and connected to the Windows or Linux platform running Oracle Java ME SDK 8.3 and that NetBeans IDE 8.1 has already been started.

Adding the Intel Galileo Gen2 Board to the Device Connection Manager

If you want to use the Intel Galileo Gen2 with NetBeans, you must first add your board to the Device Connection Manager in Oracle Java ME SDK 8.3 as follows.

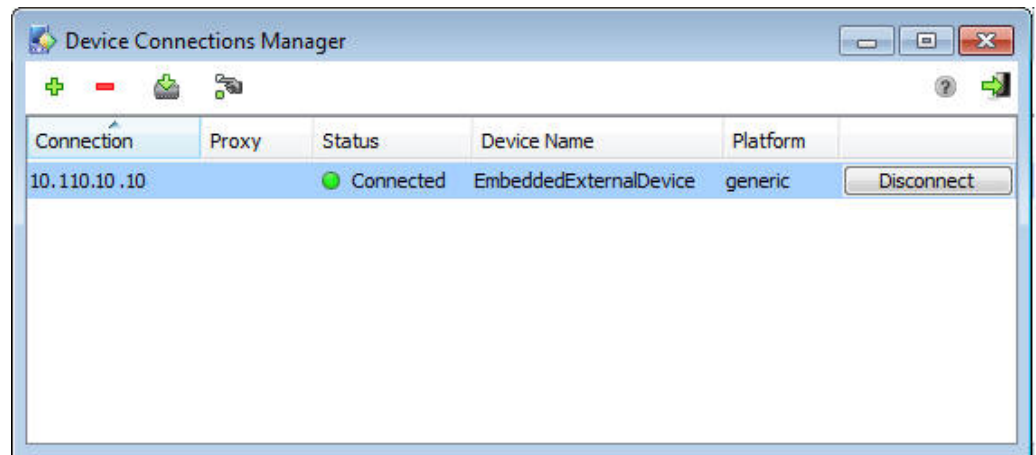
1. Ensure that the `usertest.sh` script in the `/bin` directory is running on the Intel Galileo Gen2 board.
2. Ensure that the Developer Agent program does not run on the desktop computer.
3. Start the Oracle Java ME SDK 8.3 Device Connections Manager (located in the `<SDK Installation Folder>/bin` directory) and click its icon in the system tray. A Device Connections Manager window is shown in [Figure 2-3](#).

Figure 2-3 Device Connections Manager Window



4. Click the **Add new device connection** button, ensure that the IP Address or Host Name list contains the correct IP address of the Intel Galileo Gen2 board, and click **OK**.
5. After the Intel Galileo Gen2board is registered, its IP address is listed on the Device Connections Manager list and its status is Connected as shown in [Figure 2-4](#).

Figure 2-4 *Device Connections Manager Window with Intel Galileo Gen2 Connected*



Intel Galileo Gen2Device I/O Preconfigured List

This appendix describes the proper ID and names for the various peripheral ports and buses for the Intel Galileo Gen2 board, which are accessible using the Device I/O APIs.

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

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

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

The tables use the following legend:

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

Functionality of the `galileo_pin_config` Script

The `galileo_pin_config.sh` script configures the Linux `sysfs` interface for designating the GPIO/I2C/SPI/UART protocols for different GPIO pins.

The `galileo_pin_config.sh` script resides in the `/bin` directory and must be run before running any IMlet that uses the DIO API. This script configures the kernel `sysfs` interface for the supported DIO interfaces and designates the corresponding pins as specified :

- GPIO - 6 output pins (D2,D3,D6,D7,D8,D9) and 6 Input pins (D4,D5,A0,A1,A2,A3)
- I2C - A5(SCL), A4(SDA) and dedicated SCL, SDA pins
- SPI - D13(MCLK), D12(MISO), D11,(MOSI), D10(CS)
- UART - D0(RX), D1(TX)

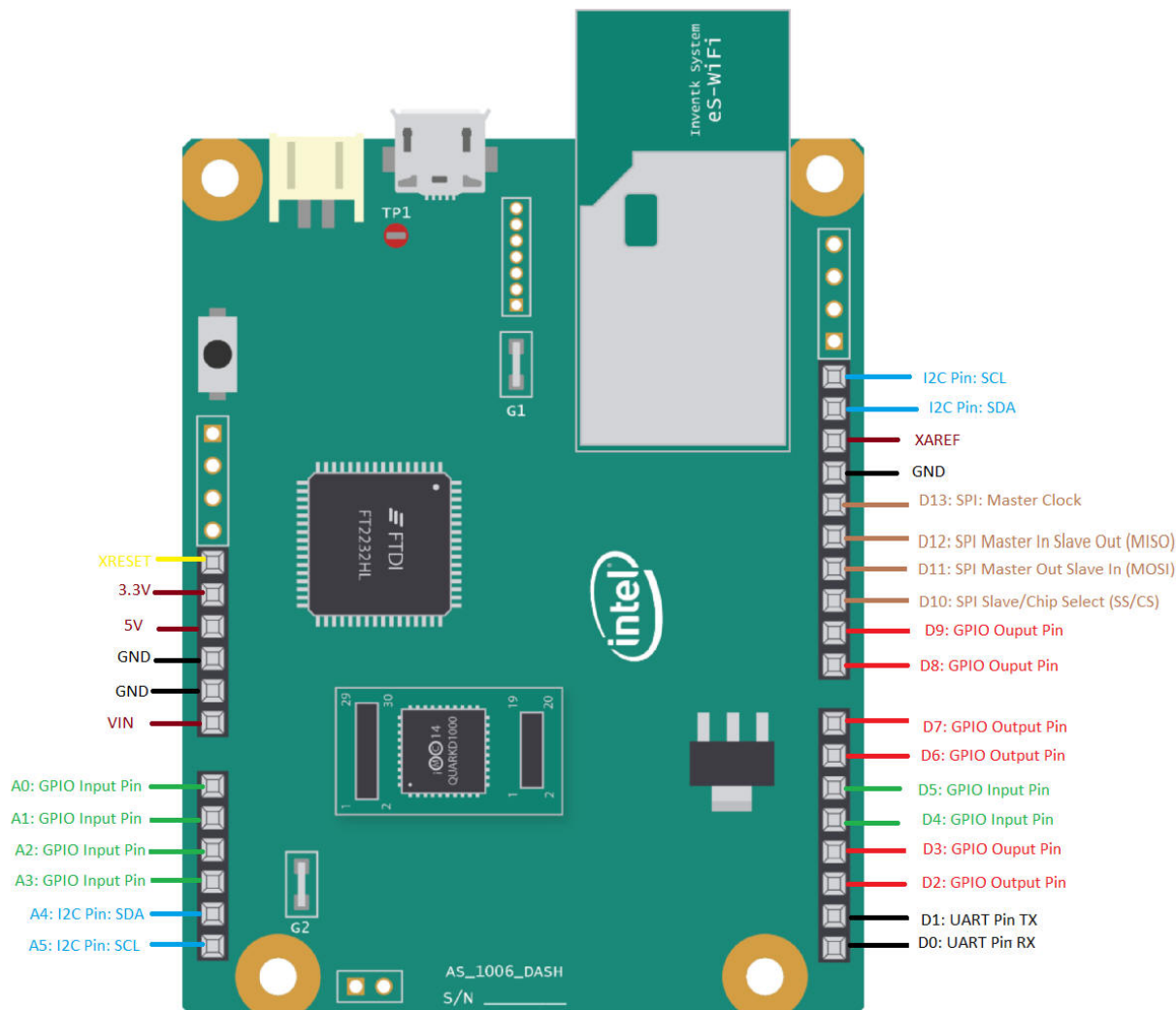
- Watchdog (software implementation)

You need to run this script only once per a boot session. Therefore it is recommended that you add this script to the `/etc/profile` or `.bashrc` to enable running the `galileo_pin_config.sh` script automatically at login time.

Reference Pin Diagram

The Reference Pin Diagram for Java ME Device I/O API on the Intel Galileo Gen2 board is shown in this section.

Figure A-1 Intel Galileo Reference Pin Diagram for Java ME Device I/O API



GPIO Pins

The following GPIO pins are preconfigured.

Device ID	Device Name	Mapped	Configuration
1	GPIO1, GPIO0.13, D2		pinNumber = 13 direction = GPIOPinConfig.DIR_OUTPUT_ONLY mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL
2	GPIO2, GPIO0.14, D3		pinNumber = 14 direction = GPIOPinConfig.DIR_OUTPUT_ONLY mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL
3	GPIO3, GPIO0.6, D4		pinNumber = 3
4	GPIO4, GPIO0.0, D5		pinNumber = 0
5	GPIO5, GPIO0.1, D6		pinNumber = 1 direction = GPIOPinConfig.DIR_OUTPUT_ONLY mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL
6	GPIO6, GPIO0.38, D7		pinNumber = 38 direction = GPIOPinConfig.DIR_OUTPUT_ONLY mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL
7	GPIO7, GPIO0.40, D8		pinNumber = 40 direction = GPIOPinConfig.DIR_OUTPUT_ONLY mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL
8	GPIO8, GPIO0.4, D9		pinNumber = 4 direction = GPIOPinConfig.DIR_OUTPUT_ONLY mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL
9	GPIO9, GPIO48.0, A0		pinNumber = 48
10	GPIO10, GPIO50.0, A1		pinNumber = 50

Device ID	Device Name	Mapped	Configuration
11	GPIO11, GPIO52.0, A2		pinNumber = 52
12	GPIO12, GPIO54.0, A3		pinNumber = 54

GPIO Default Values

Consider the following default values concerning GPIO on the Intel Galileo Gen2 board.

- Unassigned `gpio.GPIOPIN.initValue` is 0 (false).
- Unassigned `gpio.GPIOPIN.controllerNumber` is 0.
- Unassigned `gpio.GPIOPIN.direction` is `GPIOPinConfig.MODE_INPUT_PULL_DOWN`.
- Unassigned `gpio.GPIOPIN.trigger` is `GPIOPinConfig.TRIGGER_NONE`.

Configured GPIO Pins Directions

The `galileo_pin_config` script configures 12 GPIO pins as follows:

- Output pins: D2, D3, D6, D7, D8, D9
- Input pins: D4, D5, A0, A1, A2, A3, A4

Note: The specified pin directions are fixed for using the DIO API in Java ME applications. IMlets must use the GPIO pins in the specified directions otherwise the correct results are not guaranteed.

SPI

For connecting SPI devices to the Intel Galileo Gen2 board, modify the `grub.conf` file as described in [Editing the grub.conf File](#).

Consider the following implementation notes concerning SPI on the Intel Galileo Gen2board.

- SPI Pins are configured as follows:
 - D13 (Master Clock)
 - D12 (Master In Slave Out)
 - D11 (Master Out Slave In)
 - D10 (Slave Select/Chip Select)
- There are no onboard SPI devices on the Intel Galileo Gen2 board.

I2C

Device ID	Device Name	Mapped	Configuration
200	PCA9547		controllerNumber=0 address=112 clockFrequency=-1

Please note the following items about I2C on the Intel Galileo Gen2 board.

- There is one onboard I2C GPIO multiplexer switch at the address 0x70.
- The configured Pins are A4 (SDA) and A5 (SDL).

UART

The following UART devices are preconfigured:

Device ID	Device Name	Mapped	Configuration
100	ttyS0, UART0		controllerName = ttyS0 baudRate = 9600 dataBits = DATABITS_8 parity = PARITY_NONE stopBits = STOPBITS_1 flowcontrol = FLOWCONTROL_NONE
101	ttyS1, UART1		controllerName = ttyS1 baudRate = 115200 dataBits = DATABITS_8 parity = PARITY_NONE stopBits = STOPBITS_1 flowcontrol = FLOWCONTROL_NONE

Please note the following items about UART on the Intel Galileo Gen2.

- Only two UART controllers are supported on the Intel Galileo Gen2 device.

UART Default values

Consider the following default values concerning UART on the Intel Galileo Gen2 board.

- Unassigned `uart.UART.baudRate` is 9600.
- Unassigned `uart.UART.parity` is 0.
- Unassigned `uart.UART.dataBits` is 8.
- Unassigned `uart.UART.stopBits` is 1.
- Unassigned `uart.UART.flowControl` is `FLOWCONTROL_NONE`.

Structure of the Device I/O Configuration File

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

General File Data Format

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

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

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

Configuration Hierarchy

The configuration file has three sections.

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

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

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

The configs Section

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

Example A-1 Example of GPIO Pin Device Configuration

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

```
10 : {
  deviceType : gpio.GPIOPin,
  id:1,
  deviceNumber : 1,
  pinNumber : 5,
  direction : 1,
```

```

mode:4,
name :[LED_1,GPIO6.6,GPIO0]
initValue:0,
}

```

The defaults Section

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

Example A-2 Example of the `daapi_config.json` File

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

```

{
  configs: {
    13: {
      deviceType: atcmd.ATDevice,
      name: EMUL,
      properties : ["com.oracle.sms=true"]
    },
    15: {
      deviceType: uart.ModemUART,
      deviceNumber: 1,
      name: COM1,
      baudRate: 9600,
      dataBits: 7,
    },
  },
  defaults: {
    uart.UARTConfig: {
      baudRate:115200,
      parity:0,
      dataBits:8,
      stopBits:1,
      flowControl:0,
    },
  },
}

```

Glossary

access point

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

ADC

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

AMS

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

APDU

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

API

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

ARM

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

AT commands

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

AXF

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

BIP

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

CDMA

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

CLDC

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

configuration

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

DAC

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

ETSI

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

GCF

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

GPIO

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

GPIO port

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

GSM

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

HTTP

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

HTTPS

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

I2C

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

ICCID

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

IMP-NG

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

IMEI

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

IMlet

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

IMlet Suite

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

IMSI

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

ISA

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

JAD file

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

JAR file

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

Java ME platform

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

JCP

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

JSR

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

JVM

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

KVM

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

LCDUI

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

MIDlet

An application written for MIDP.

MIDlet suite

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

MIDP

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

MSISDN

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

MVM

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

obfuscation

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

optional package

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

preverification

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

Profile

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

Provisioning

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

Pulse Counter

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

Push Registry

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

RISC

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

RL-ARM

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

RMI

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

RMS

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

RTOS

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

RTSP

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

SCWS

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

SD card

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

SIM

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

Slave mode

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

smart card

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

SMS

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

SMSC

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

SOAP

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

SPI

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

SSL

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

SVM

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

task

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

TCP/IP

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

Terminal Profile

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

UART

Universal Asynchronous Receiver/Transmitter. A piece of computer hardware that translates data between serial and parallel formats. It is used to facilitate

communication between different kinds of peripheral devices, input/output streams, and embedded systems, to ensure universal communication between devices.

UICC

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

UMTS

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

URI

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

USAT

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

USB

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

USIM

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

WAE

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

WAP

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

watchdog timer

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

WCDMA

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

WMA

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

XML Schema

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