Oracle® Java ME Embedded

Getting Started Guide for the Windows Platform

Release 8 EA 2 for NetBeans on Windows

E48511-01

February 2014

This book describes how to use Oracle Java ME SDK 8 EA 2 to develop embedded applications, using the NetBeans 8.0 Beta Integrated Development Environment, on a Windows 7 platform.

Beta Draft

ORACLE

Oracle Java ME Embedded Getting Started Guide for the Windows Platform, Release 8 EA 2 for NetBeans on Windows

E48511-01

Copyright © 2012, 2014, Oracle and/or its affiliates. All rights reserved.

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 failsafe, 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 on 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. 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.

This documentation is in pre-production 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.

Contents

4 Using the External Events Generator

5 Working With Devices

6 Using Sample Applications

A Using the Command Line Emulator

B Installation and Runtime Security Guidelines

List of Figures

Preface

This book describes how to use the Oracle Java ME SDK 8 Early Access (EA) 2 platform to quickly develop embedded applications on the NetBeans 8.0 Beta, Integrated Development Environment (IDE). The Oracle Java ME SDK 8 EA 2 platform contains a complete implementation of the Oracle Java ME Embedded 8.0 EA 2 software runtime.

Together, these products support embedded software development on:

- Windows 32 platform in emulation (this guide)
- Qualcomm Internet of Everything (IoE) platform (*Oracle Java ME Embedded Getting Started Guide for the Reference Platform (Qualcomm Orion IoE))*
- Raspberry Pi (*Oracle Java ME Embedded Getting Started Guide for the Reference Platform (Raspberry Pi))*

Audience

This document is intended for embedded developers who want to develop applications on Oracle Java ME SDK 8 EA 2 on a Windows 7 platform.

Documentation Accessibility

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

Access to Oracle Support

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

Operating System Commands

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

Shell Prompts

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

Conventions

The following text conventions are used in this document:

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

1

Before You Begin

The Oracle Java ME SDK 8 EA 2 platform is a sophisticated and useful tool for programmers who want to develop embedded applications. The Oracle Java ME SDK 8 EA 2 platform can be used with NetBeans 8.0 Beta (older versions, such as NetBeans 7.3.1, are not supported).

> **Note:** The Oracle Java ME Embedded 8.0 EA runtime is implemented inside Oracle Java ME SDK 8 EA 2 software. So, there is no need for you to independently install Oracle Java ME Embedded 8.0 EA runtime.

This chapter provides information you need to ensure that your Microsoft Windows 7 (32-bit or 64-bit) platform is correctly set up for working with Oracle Java ME SDK 8 EA 2. Windows 7 must include the most recent Microsoft service packs.

Installing the Java SE Platform

To properly run the Oracle Java ME SDK 8 EA 2 software and its associated Tools, you must have Java Platform, Standard Edition (Java SE), Version 7, Update 45 (or later) installed on your computer.

This guide assumes you have already installed the Java SE platform. If you have not, you can download it from the following location:

`http://www.oracle.com/technetwork/java/javase/downloads`

The Java SE platform must also be in your PATH.

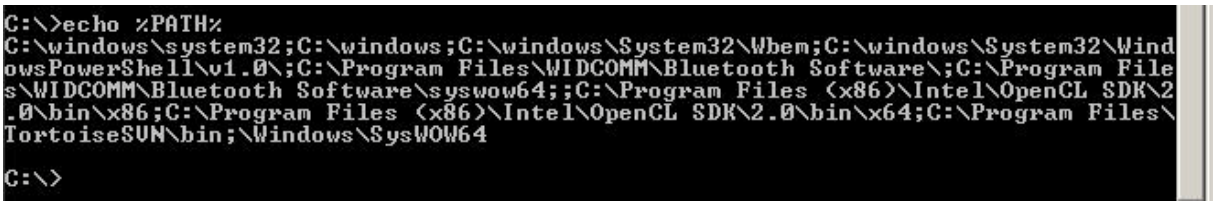
Setting and Verifying Your Java SE PATH

To verify if Java SE platform is set in your PATH:

1. In the Windows command line, type the following, as shown in Figure 1–1.

   C:\>**echo %PATH%**

Figure 1–1 Output from the echo PATH Command



2. If you do not see Java SE included in the PATH output, you need to add Java SE to your PATH.

   Note: Setting the PATH may require using a Windows short name. To see top-level Windows short names, type C:\>**dir /x**

3. Set the Java SE variable, JDK_DIR:

   C:\>**set JDK_DIR=C:\Program Files\Java\jdk1.7.0_45**

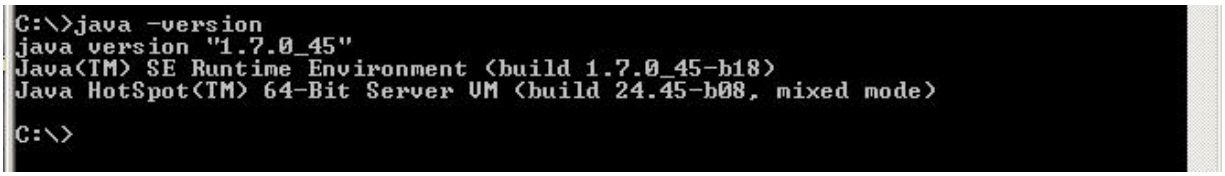
4. Add JDK_DIR to your PATH:

   C:\>**set PATH=%PATH%;%JDK_DIR%\bin**

5. To verify the version of your Java SE platform, type the following, as shown in:

   C\:>**java -version**

Figure 1–2 Output from the java -version Command



The version number shown in the output should be version 1.7.0_45 or higher.

Installing the Oracle Java ME SDK 8 EA 2 Platform

Follow these steps to install the Oracle Java ME SDK 8 EA 2.

1. If you have previously installed an earlier version of Oracle Java ME SDK, uninstall the previous version as shown below.
   - If you have Oracle Java ME SDK data to save, copy it to a safe location before continuing.
   - In the Windows system tray, right click the emulator icon and choose Exit.
   - From the Windows Programs menu, select the previous version and choose Uninstall from the submenu. The Installer opens.
   - On the first page check the option to remove the user data directory.
   - Follow the prompts.
2. Download the Oracle Java ME SDK 8 EA 2 from:

   http://www.oracle.com/technetwork/java/javame/javamobile/download/sdk
3. Double-click the executable. When the installer starts, follow the prompts.

Installing and the Starting NetBeans 8.0 Beta

If you do not already have the NetBeans 8.0 Beta IDE installed on your system, you can download it here:

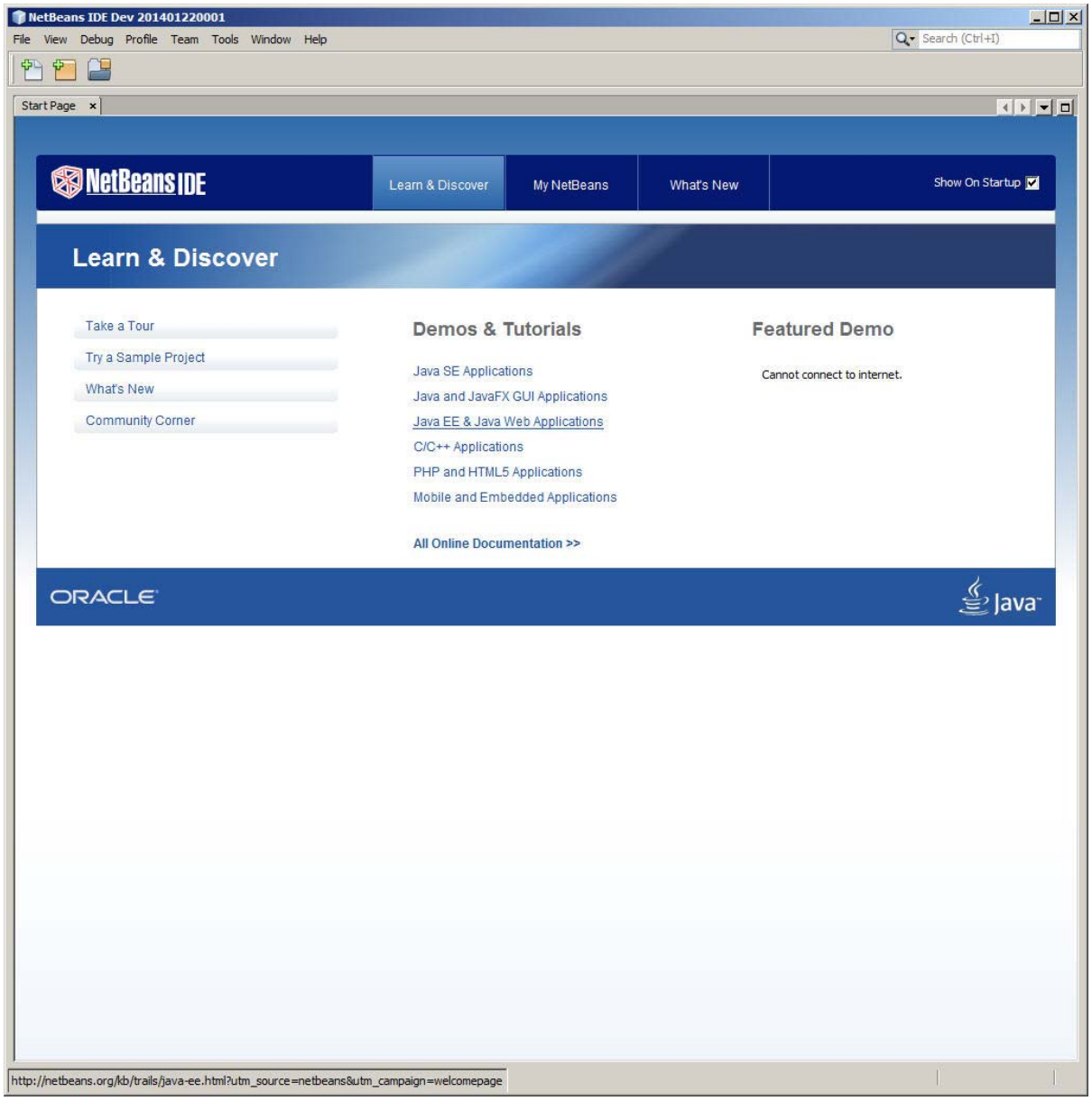
http://bits.netbeans.org/download/trunk/nightly/2014-02-03_00-01-07/

Do the following.

1. Download the version of NetBeans 8.0 Beta appropriate for Java ME (ALL).
2. When the download is complete, double click the NetBeans 8.0 Beta executable.
3. This starts the NetBeans 8.0 Beta installer. Follow the directions. When the installation is complete, you see the NetBeans 8.0 Beta icon on your Desktop.
4. Double click the icon to start NetBeans 8.0 Beta, as shown in Figure 1–3.

Note: The NetBeans 8.0 Beta download version described here (version 201402030001) is the tested and approved version for working with Oracle Java ME SDK 8 EA 2. Later versions of NetBeans may also work, but there is more risk using an untested version.

Figure 1–3 The NetBeans 8.0 Beta Start Screen



For more information on working with the NetBeans 8.0 Beta, see Chapter 2, "Creating an ME SDK 8 Sample Project."

2

Creating an ME SDK 8 Sample Project

This chapter describes the NetBeans 8.0 Beta integrated development environment. NetBeans 8.0 Beta provides a rich, visual environment for developing embedded applications and numerous tools to improve the programming process.

Oracle Java ME SDK provides two plugins for working with NetBeans 8.0 Beta:

- Java ME SDK Tools plugin
- Java ME SDK Demos plugin

The Java ME SDK Demos plugin is optional, but recommended.

Downloading Oracle Java ME SDK Plugins

To download the Oracle Java ME SDK 8 EA 2 Plugins file for NetBeans (`oracle-jmesdk-8-0-ea2-nb-plugins.zip`) go to the following location:

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

Installing Oracle Java ME SDK 8.0 EA Plugins

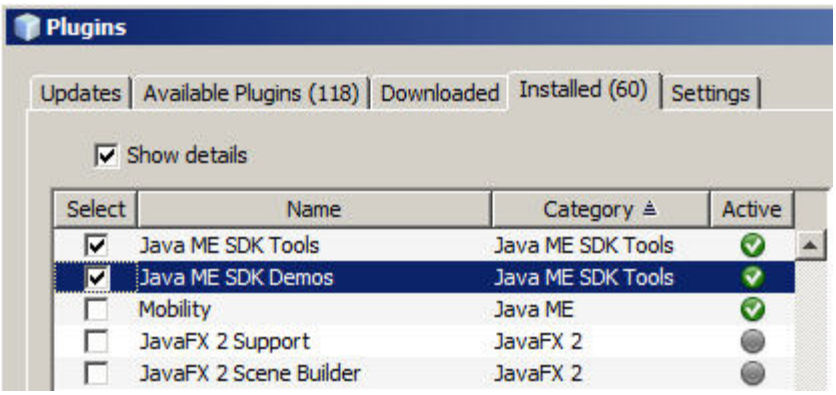
To install the NetBeans Plugins:

1. Extract the contents of the NetBeans Plugins file to a directory on your local machine. Make note of the location.
2. **Open** the NetBeans Plugins Manager. Select:

   Tools > Plugins
3. Uninstall previous plugins.
   - Go to the **Installed** tab and click **Show details**.
   - Check `Java ME SDK Tools` and `Java ME SDK Demos`, as shown in Figure 2–1.

   Note: If the previous plugins have already been uninstalled, go to Step 4.

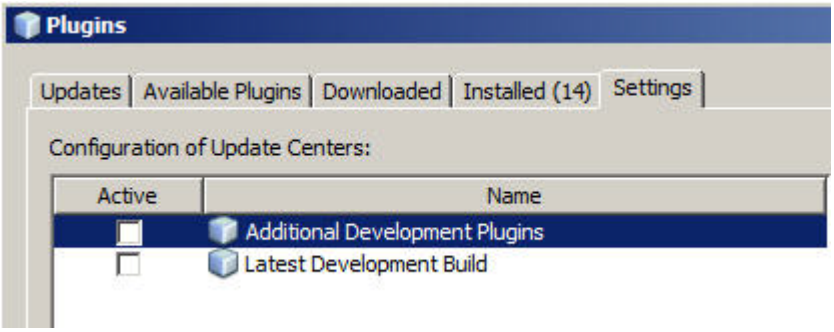
Figure 2–1 NetBeans Plugins Manager Window



- Click **Uninstall**.
- Restart when requested.

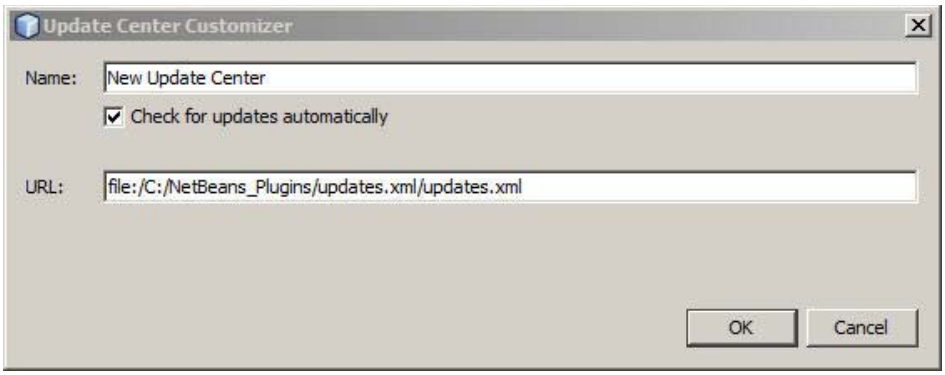
4. If not already open, re-open the Plugins Manager and click the **Settings** tab. Make sure that Additional Development Plugins and Latest Development Build are unchecked, as shown in Figure 2–2.

Figure 2–2 The Settings Tab



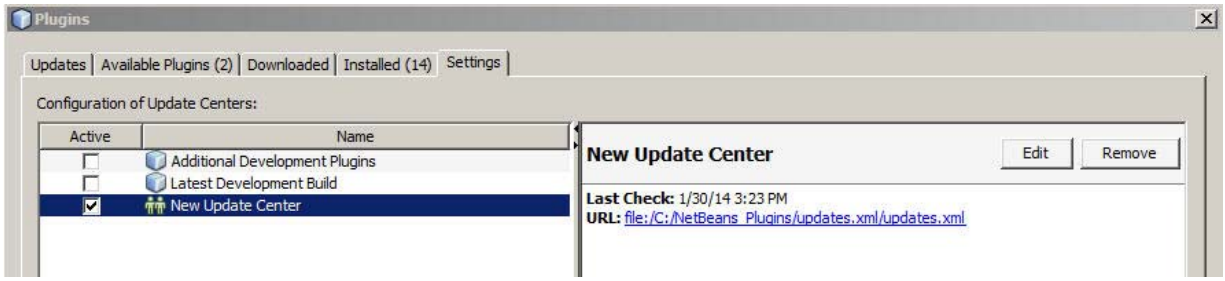
5. Click the **Add** button in the lower right. This displays the Update Center Customizer box.
6. In the Update Center Customizer box, do the following, as shown in Figure 2–3:
   - In the Name field, enter **New Update Center**.
   - Select the **Check for updates automatically** box.
   - In the URL field, use the `file` command to point to the location where you have unzipped your plugins. For example:
   - `file:/C:/My_Update_Center_Plugins/updates.xml`

Figure 2–3 Adding Plugins



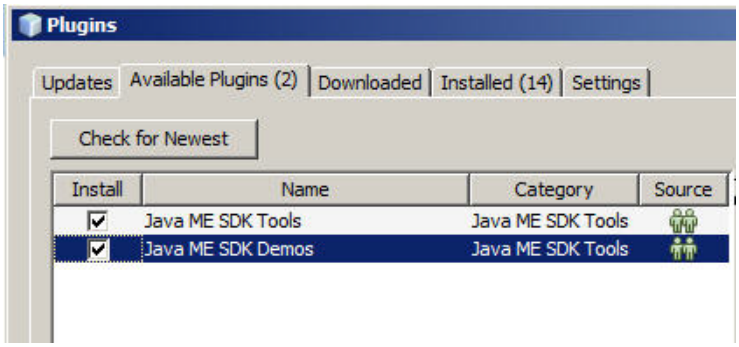
7. Click **OK**. This adds the New Update Center to the **Settings** tab.
8. Select the **New Update Center**, as shown in Figure 2–4.

Figure 2–4 Selecting the New Update Center



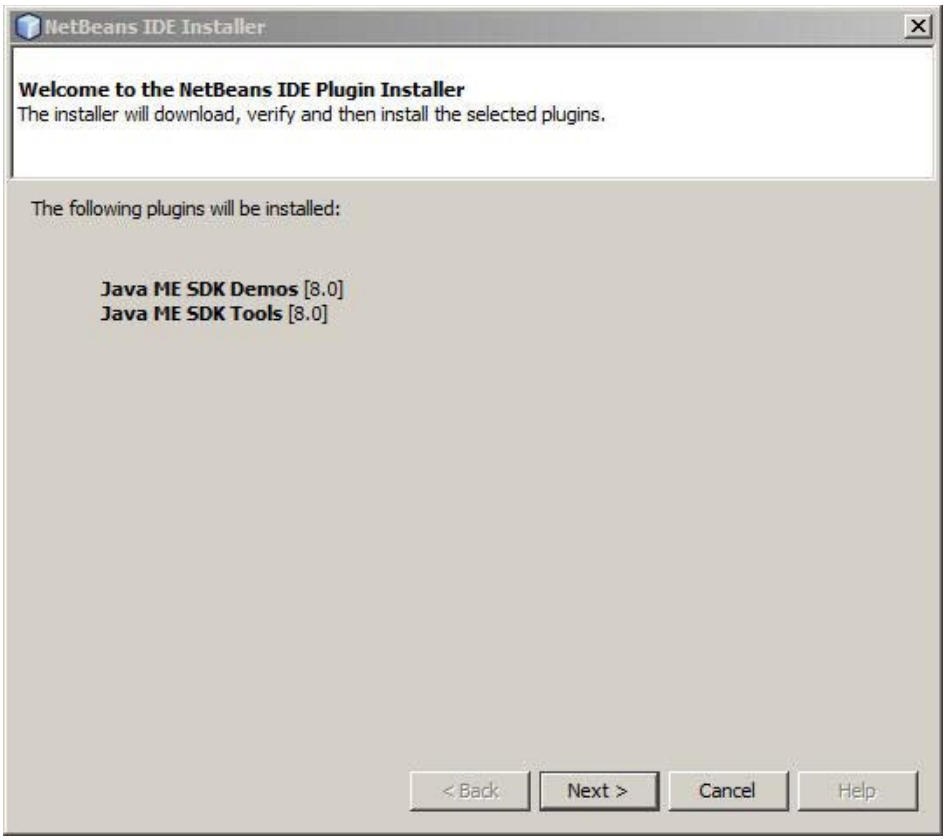
9. Go to the **Available Plugins** tab and select Java ME SDK Tools and Java ME SDK Demos, as shown in Figure 2–5, and click **Install**.

Figure 2–5 Selecting Available Plugins



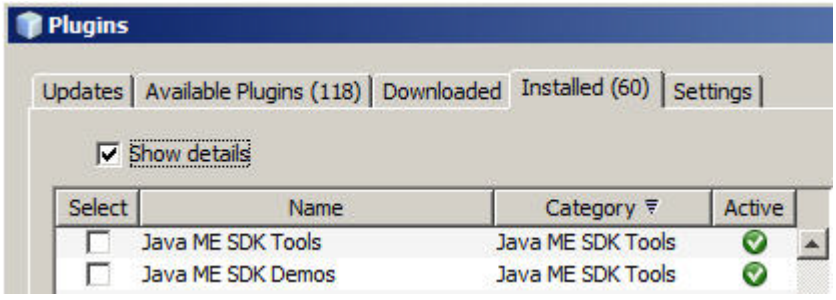
10. When the NetBeans IDE Installer screen is displayed, as shown in Figure 2–6, click **Next**.
    - Accept the license terms and click **Install**.
    - If additional Validation screens appear, click **Continue**.

Figure 2–6 The NetBeans IDE Plugin Installer



11. Click **Finish** to restart NetBeans 8.0 Beta.
12. When NetBeans 8.0 Beta has restarted, select **Tools > Plugins** to display the Plugins screen.
13. In the **Installed** tab, check **Show details** and click **Category** to sort the plugins.
    - Find the `Java ME SDK Tools` and `Java ME SDK Demos` plugins in the list.
    - Make sure the plugins you have installed are Active (you should see a green check mark), as shown in Figure 2–7.
    - If the Oracle Java ME SDK 8 EA 2 plugins are not Active, check the Select boxes for the plugins and click **Activate**.

Figure 2–7 Activated Plugins



14. When your Oracle Java ME SDK 8 EA 2 plugins are Active, click **Close**.

Creating a Sample IMlet File

In this section, you create a sample IMlet file, `IMletDemo.java`, from the code provided in Example 2–1. This IMlet file is used in the next section, "Creating a New Project".

1. Copy the code shown in Example 2–1 into a text file. Use Notepad rather than WordPad, to avoid any unneeded extra characters.
2. Name the file `IMletDemo.java` and Save. Set the file aside for now.

Example 2–1 Code for the Sample IMletDemo.java Project in NetBeans 8.0 Beta

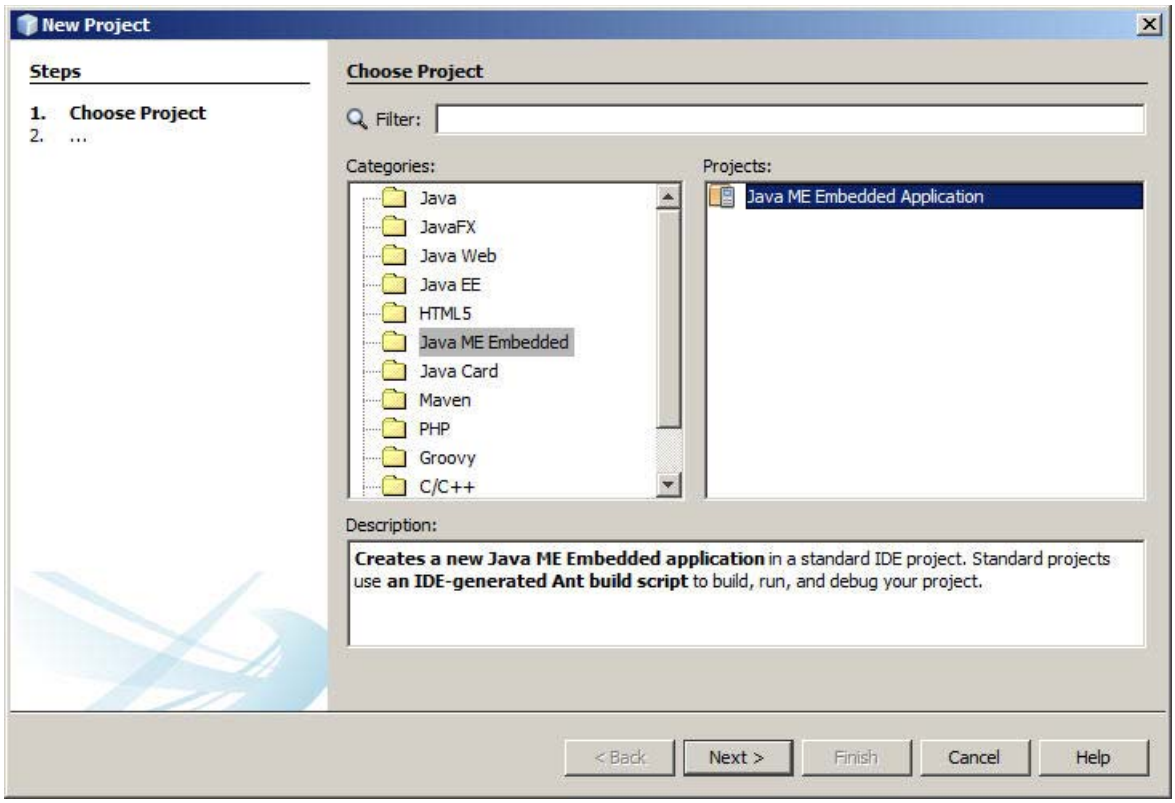
```
package javameapplication1;
import javax.microedition.midlet.MIDlet;

public class IMletDemo extends MIDlet {

     boolean bFirst = false;

     public void startApp() {

            try {
                // Perform startup operations
            } catch (Exception ex) {
                ex.printStackTrace();
                return;
            }

            bFirst = true;
              System.out.println("IMlet Demo is already started...");
          // Start program here

}
    public void pauseApp() {
         //  Pause the application
    }

    public void destroyApp(boolean unconditional) {
       // Close all resources that have been opened

    }


}
```

Creating a New Project

This section walks you through creating a new embedded project using the Oracle Java ME SDK 8 EA 2 and NetBeans 8.0 Beta platforms.

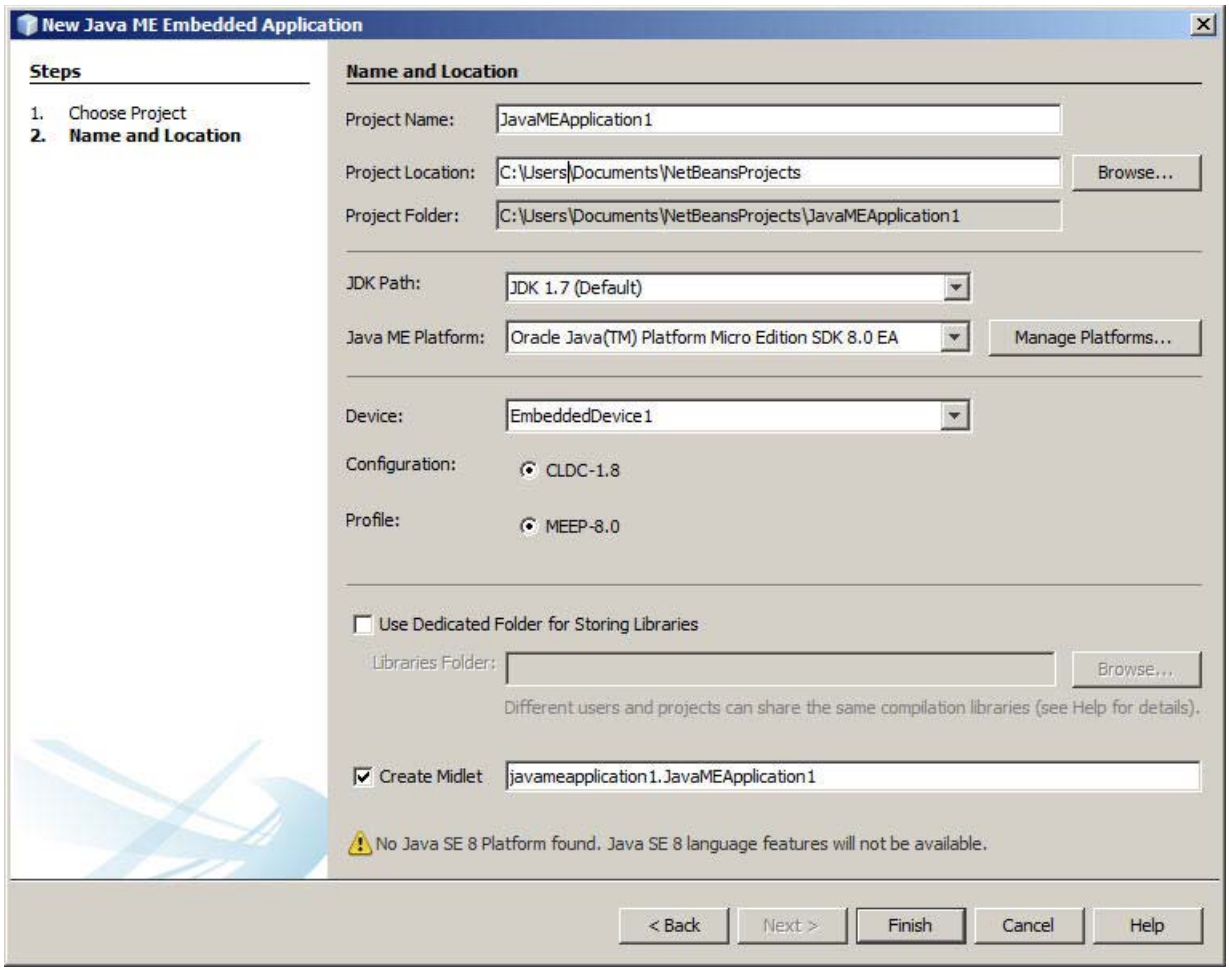
1. Choose **File > New Project**. The **New Project** dialog box appears.
2. Choose Java ME Embedded from the Categories list and Java ME Embedded Application from the Projects list, as shown in Figure 2–8.

Figure 2–8 The New Project Screen



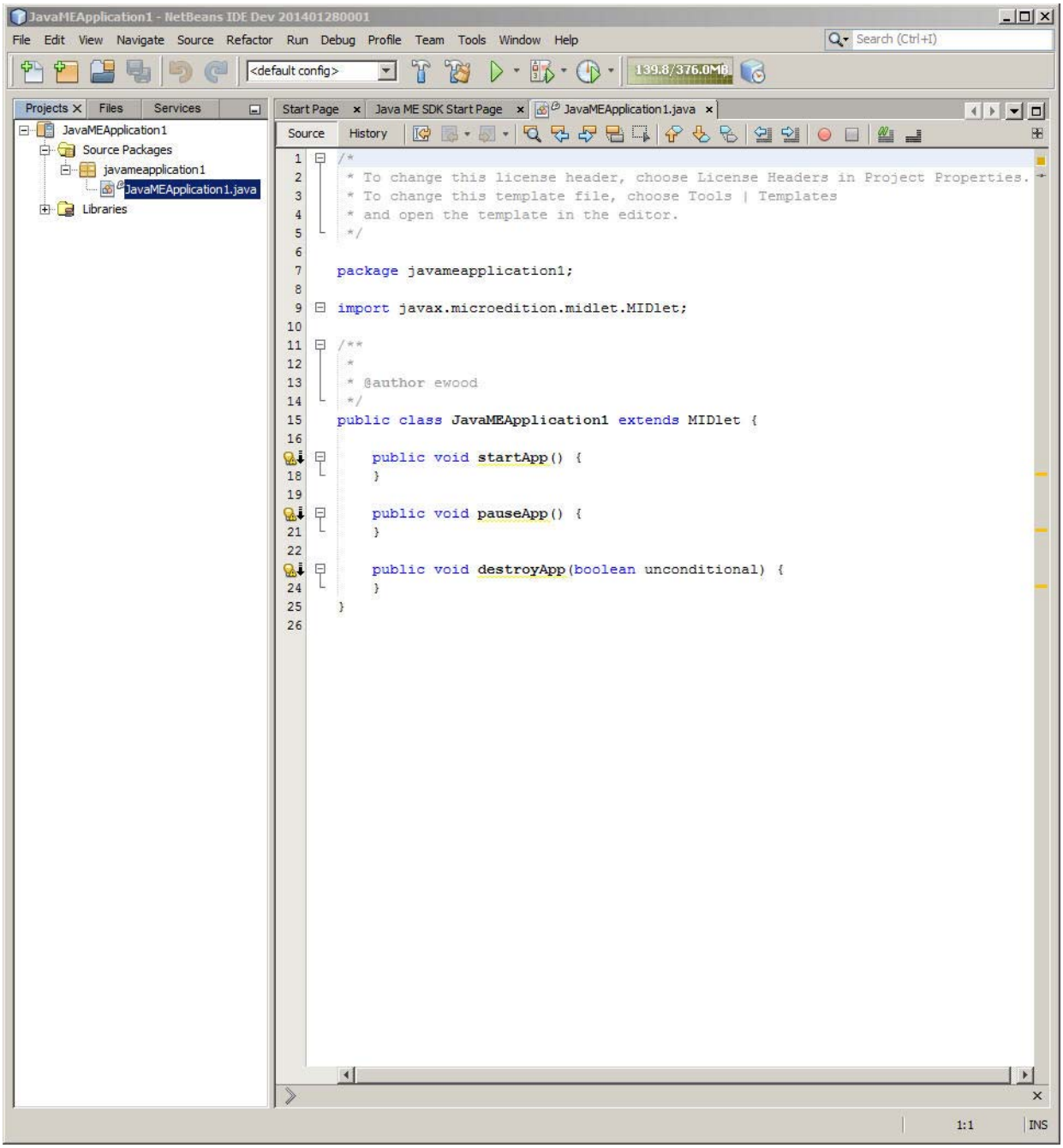
3. Press the **Next** button. This shows the New Java ME Embedded Application window, as shown in Figure 2–9.

Figure 2–9 The New Java ME Embedded Application Screen



4. Accept the defaults and click **Finish**.
5. The NetBeans 8.0 Beta New Project screen is displayed for the project you have just created, containing a minimal default code sample, as shown in Figure 2–10.

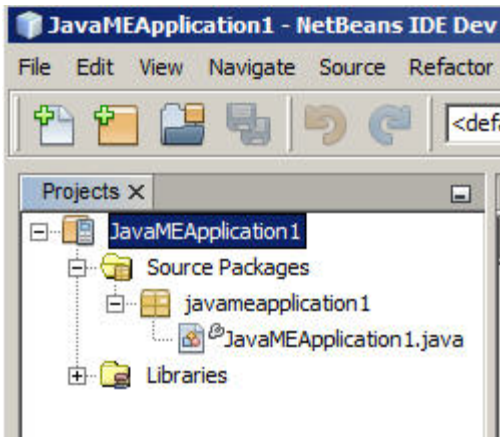
Figure 2–10 Sample Code for The Newly-Created Java ME Application



Including Your Sample IMlet Code

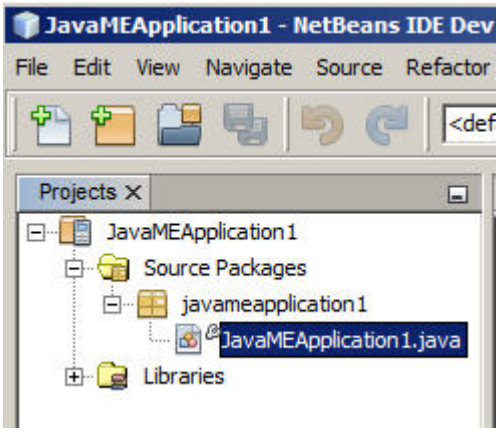
Now you can update the generic project with the sample code you created earlier in Creating a Sample IMlet File. The NetBeans 8.0 Beta Projects window is visible in Figure 2–11.

Figure 2–11 The NetBeans Projects Window



1. Right click to select JavaMEApplication1 in the Projects window, as shown in Figure 2–12 and select Properties from the drop-down menu.

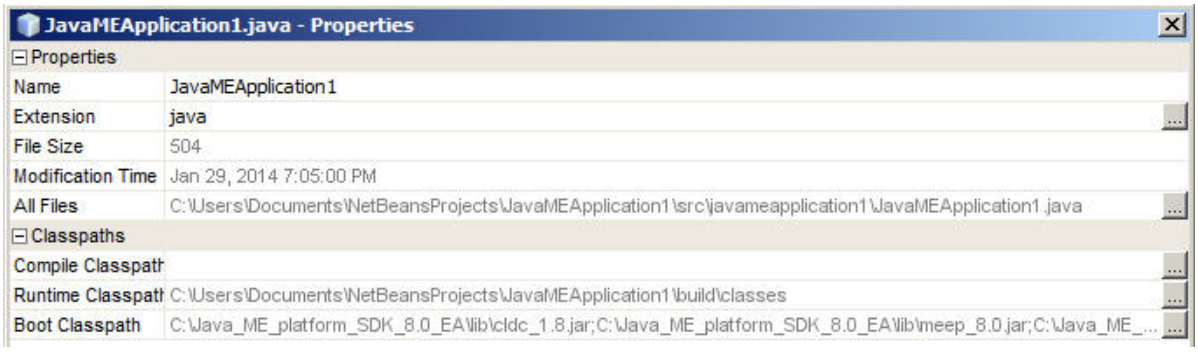
Figure 2–12 The JavaMEApplication1.java Project



2. This displays the JavaMEApplication1 Properties window.

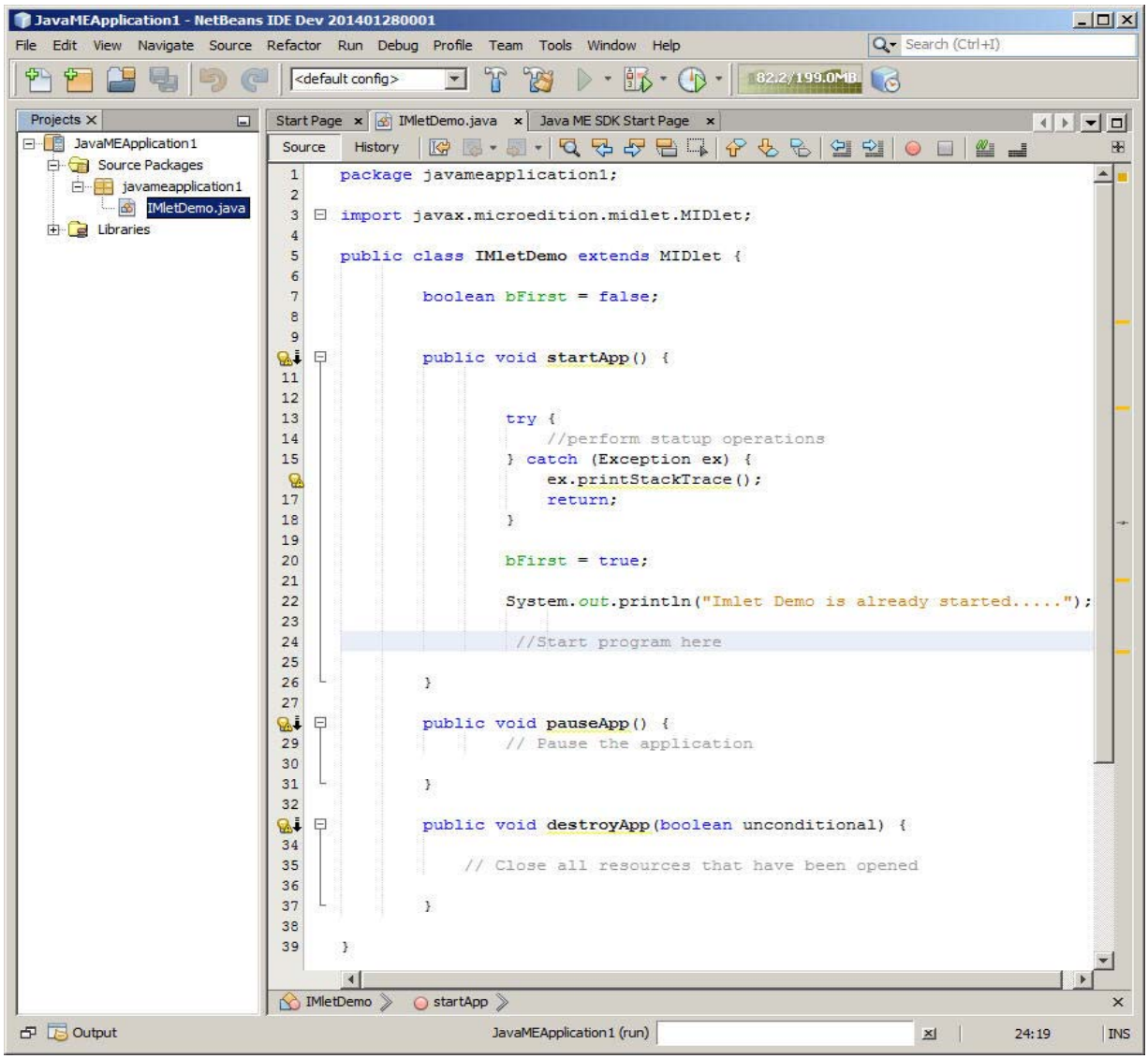
   To see the path to the location of the JavaMEApplication1 file, look in the All Files line. Make a note of this path, as shown in Figure 2–13.

Figure 2–13 The JavaMEApplication1.java Properties Screen



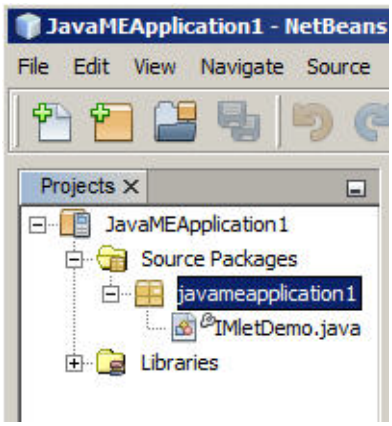
3. Take the `IMletDemo.java` file you created in Creating a Sample IMlet File and copy it into the `javameapplication1` directory in the file path you noted.
4. Delete the file `JavaMEApplication1.java` or remove it from the directory.
5. This changes the content of the NetBeans 8.0 Beta Project window to the content in the `IMletDemo.java` file, as shown in Figure 2–14. (If it does not display, double-click `IMletDemo.java` in the Projects window.)

Figure 2–14 The NetBeans 8.0 Beta Project Window with the IMletDemo Project



6. Right click on the package name `javameapplication1` in the Projects window, as shown in Figure 2–15.

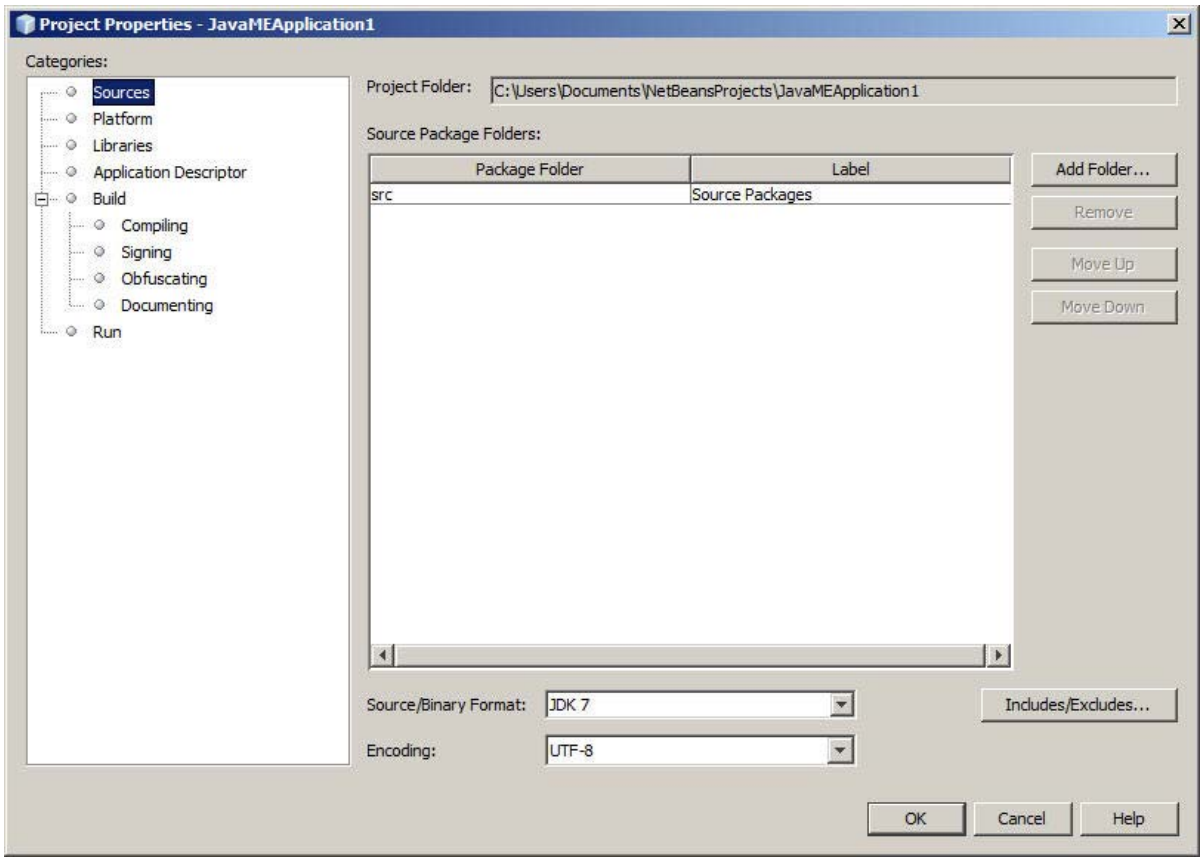
Figure 2–15 Selecting the javameapplication1 Package



7. Select **File > Project Properties (JavaMEApplication1)**.

   This displays the JavaMEApplication1 Properties box, as shown in Figure 2–16.

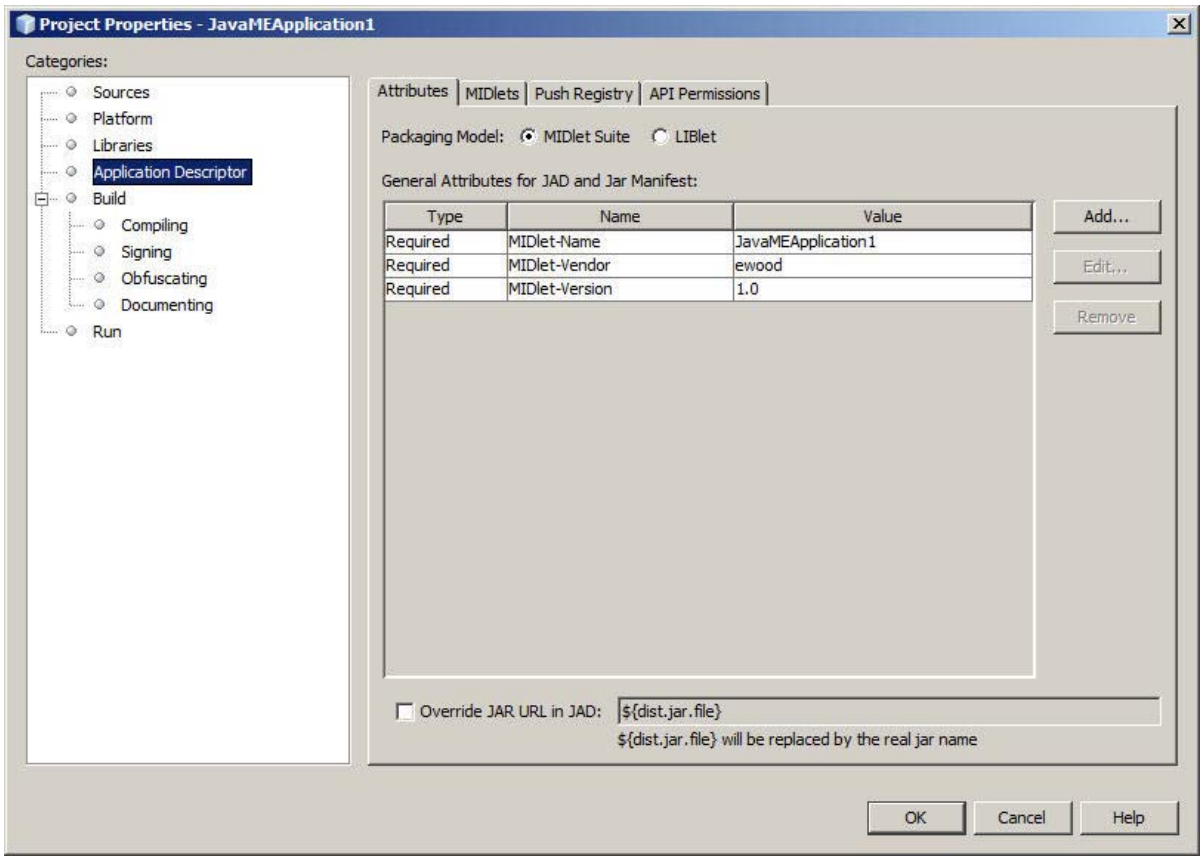
Figure 2–16 The JavaMEApplication1 Properties Box



8. Select the **Application Descriptor** category.

   This displays the Application Descriptor window, with attributes of the JAD and Jar Manifest for the default MIDlet suite, as shown Figure 2–17.

Figure 2–17 The Application Descriptor Window with MIDlet Manifest Settings

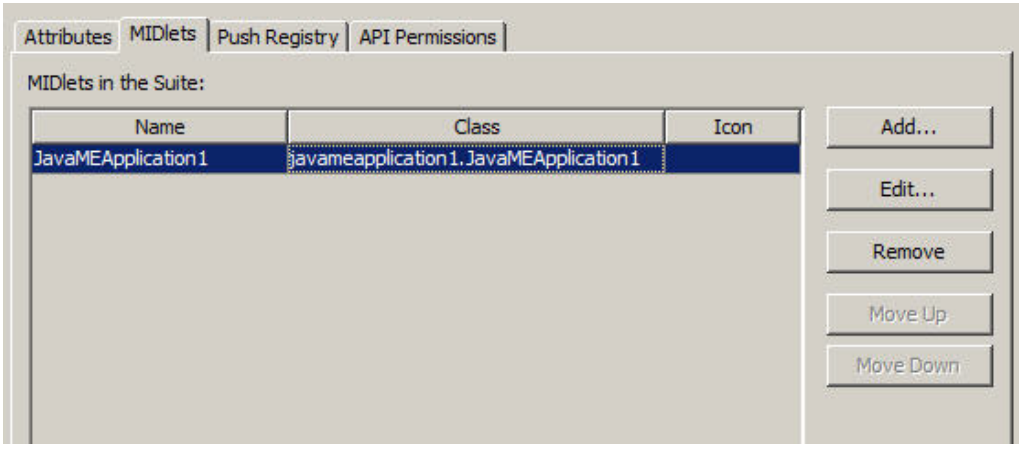


9. Select the **MIDlets** tab. The Class name in the MIDlet row is flagged red, as shown in Figure 2–18.

Figure 2–18 The MIDlet Class Name

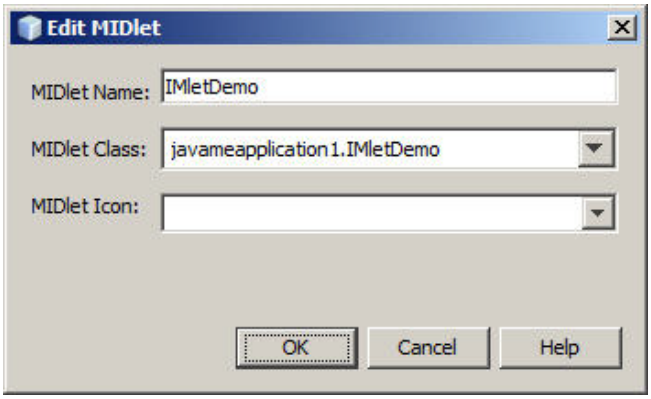
10. Select the MIDlet row. This flags the row dark blue and activates the **Edit** button, as shown in Figure 2–19.

Figure 2–19 The MIDlets Tab with the Edit Button Active



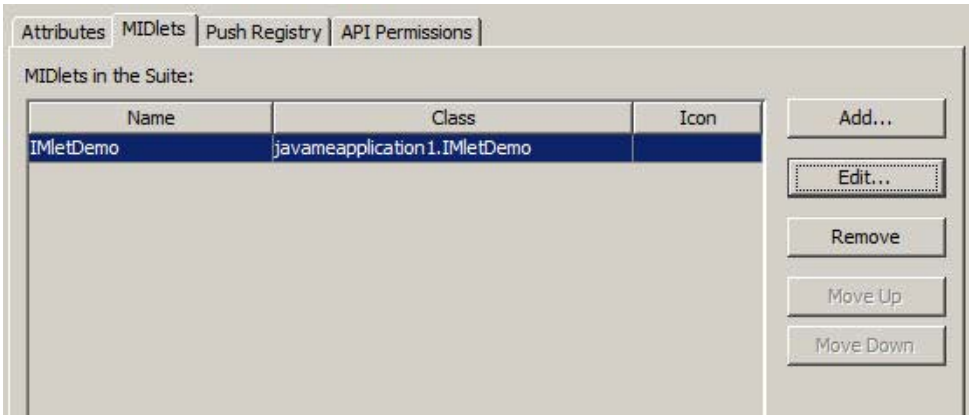
11. Press the **Edit** button to display the **Edit MIDlet** dialog box.
12. Use the drop-down list to change the `MIDlet` class to `javameapplication1.IMletDemo`, as shown in Figure 2–20, and press **OK**.

Figure 2–20 The Edit MIDlet Dialog Box



13. The MIDlet Class is now updated in the **MIDlets** tab., as shown in Figure 2–21. Press OK to close the Properties window.

Figure 2–21 The Updated MIDlet Properties Class

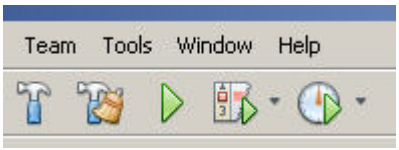


The MIDlet Properties window, showing the MIDlet updated to `javameapplication1.IMletDemo`.

****

14. Clean and build the `JavaMEApplication1` project by clicking on the hammer-and-broom icon in the NetBeans 8.0 Beta toolbar, as shown in Figure 2–22, or by selecting **Run > Clean and Build Project (JavaMEApplication1)**.

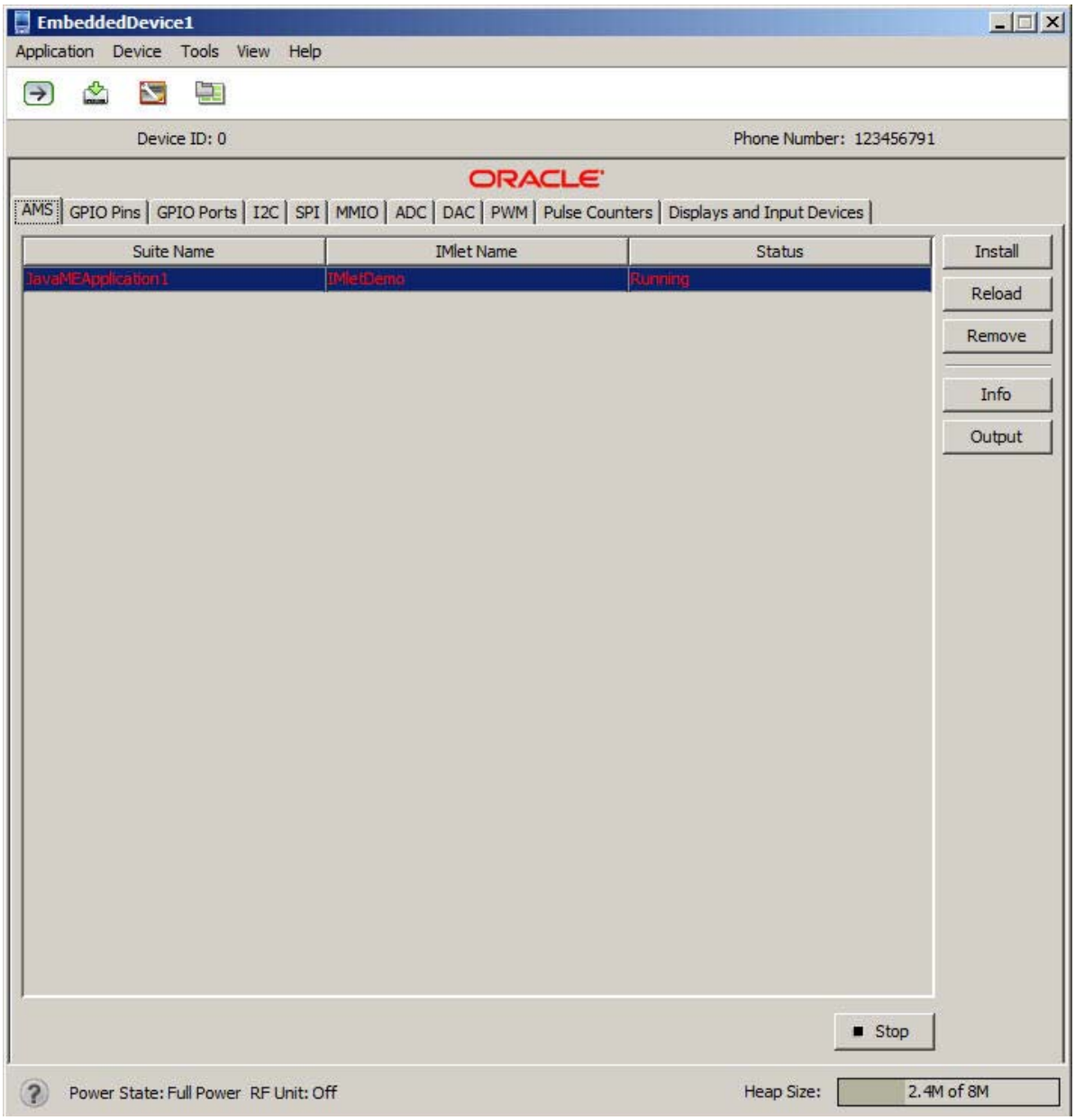
Figure 2–22 The NetBeans 8.0 Beta Toolbar and Menus



15. Run the newly cleaned and built `JavaMEApplication1` project by selecting the green right-arrow icon in the NetBeans 8.0 Beta toolbar or by selecting **Run > Run Project (JavaMEApplication1)**.

    When the Run is successful, the EmbeddedDevice1 emulator starts with the JavaMEApplication1 Suite running, as shown in Figure 2–23.

Figure 2–23 The Running EmbeddedDevice1 Emulator Window



3

Using the Emulators

The Oracle Java ME SDK 8 EA 2 embedded emulation environment provides you with a platform for testing and running Oracle Java ME Embedded Profile (MEEP) IMlet suites without having to install those IMlet suites onto an embedded device.

This is done using two default embedded emulators (`EmbeddedDevice1` and `EmbeddedDevice2`). These emulators do not represent a specific device, but provide a correct implementation of the APIs for this platform.

The `Qualcomm_IoE_Device` emulator, which is also described in this chapter, provides an emulation of the Qualcomm Orion IoE embedded hardware device. For more information, see the Oracle Java ME Embedded *Getting Started Guide for the Reference Platform (Qualcomm Orion IoE).*

This chapter describes how to run and use the Oracle Java ME SDK 8 EA 2 embedded emulators.

Launching the Emulator

If the Embedded emulator is not displayed when you finish building the sample project described in Chapter 2, it can be launched from the Windows command line. For more information, see Appendix A, "Using the Command Line Emulator."

The Emulator's Main Screen

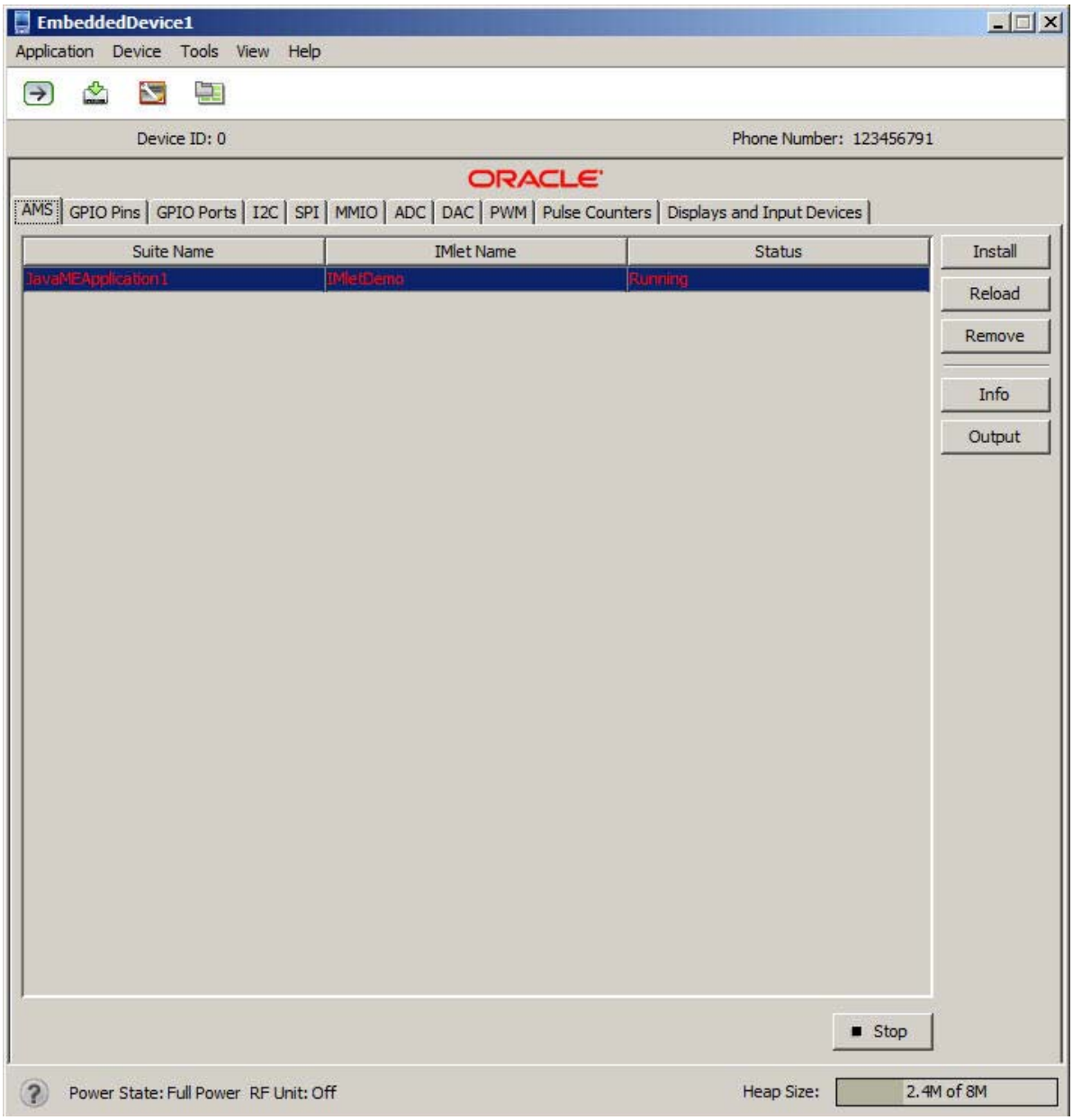
The Oracle Java ME SDK 8 EA 2 Embedded emulator provides several ways to interact with an emulated device (for example, EmbeddedDevice1), as shown in Figure 3–1.

- The **Application** menu (and associated icons) allow you to install and run IMlet suites.
- The **Tools** menu allows you to manage a Landmark Store or file system.
- The External Events Generator icon (and **Tools** menu option) allows you to configure pins, ports, interfaces, and other customizable parts.
- The **View** menu allows you to display log information and other output in a console window.
- The **Help** menu allows you to display context-sensitive help to get more information about the emulator screen and functions.

Figure 3–1 The Oracle Java ME SDK 8.0 EA Emulator Toolbar and Menu Items

When the Oracle Java ME SDK 8 EA 2 emulator starts, it appears with the Application Management System (AMS) tab selected as the default, as shown in Figure 3–2.

Figure 3–2 The Oracle Java ME SDK 8 EA 2 Emulator (AMS Tab) Default Screen



Device Name. Shown in the upper left window frame.

Menus. The Menus available on the Oracle Java ME SDK 8 EA 2 emulators are:

- **Application** - Allows you to install and run IMlet suites.
- **Device** - Allows you to observe Wireless Messaging (WMA) messages addressed to the device.
- **Tools** - Allows you to manage access points, landmarks, the file system, launch the External Events Generator.
- **View** - Allows you to view output from a running application, logging information from the emulator, and to define desktop and exit behavior.
- **Help** - Allows you to find context-specific emulator information.

Toolbar. The Toolbar provides shortcuts for the following operations:

- **Run IMlet Suite**
- **Install IMlet Suite**
- **Launch the External Events Generator**
- **Emulator window always on top**

> **Note:** When the Device Manager detects an external embedded device, only the Application, View, and Help menus and the Run IMlet Suite, Install IMlet Suite, and Emulator window always on top icons in the Toolbar are available in the device display.

Device ID. Numerical identifier that is unique for each device.

Phone Number. A number used by the emulator to send messages to itself for testing purposes.

Display Panel and **Tabs**. The display panel displays fields and information dependent on the selected tab:

- **AMS**. The AMS (Application Management System) tab displays the name of an IMlet suite, the IMlet, and the status of the IMlet suite, for all installed IMlet suites. You can select a suite and perform one of the following operations by clicking the corresponding button to the right of the display:
  - **Install** - Specify the location of the IMlet suite path or URL and the security domain and click OK to load the IMlet suite.
  - **Reload** - Reload the application.
  - **Remove** - Remove the application.
  - **Info** - Provides information about the IMlet suite.
  - **Output** - Opens a dialog box displaying output information about the application
- **GPIO Pins**. The General Purpose Input Output (GPIO) Pins tab displays which pins are selected to output and their high or low values.
- **GPIO Ports**. The GPIO Ports tab displays a list of ports, which ports are selected to output, and their maximum and current values.
- **I2C**. The Inter-Integrated Circuit (I2C) tab displays information for the selected slave device and data sent to a master device and data received from a master device.

- **SPI**. In the default SPI implementation, buffered written data can be read from the Slave named SPI. If you have created a custom implementation with the Custom Device Editor, the Slave drop down list might have additional slaves.
- **MMIO**. The memory-mapped I/O tab displays information for the current device.
- **ADC**. The Analog-to-Digital Converter tab displays the current channel, converter number, sampling intervals, minimum and maximum values, and other information.
- **DAC**. The Digital-to-Analog Converter tab displays the current channel information, converter characteristics, and a graphic display of signal characteristics with the x-axis showing the digital input and the y-axis showing the analog output.
- **PWM**. The Pulse Width Modulation tab displays the amount of electrical power flowing to a device.
- **Pulse Counters**. The Pulse Counters tab displays the ID, counter name, counter number, counter type, and the pins to which the counters are bound.
- **Displays and Input Devices**. The Displays and Input Devices tab displays color and other settings for an emulated device.

Emulator Status Bar. Information about the current system state is shown in the status bar at the bottom of the emulator window. Also shown is the memory indicator showing used and total memory.

Run/Stop Button. A toggle button that can be used to start and stop an IMlet suite.

The AMS Tab

The first tab in the emulator, the AMS tab, displays which Oracle Java ME SDK 8 EA 2 applications are installed, running, or stopped.

You can use the buttons in the AMS tab, as shown in Figure 3–3, to do the following:

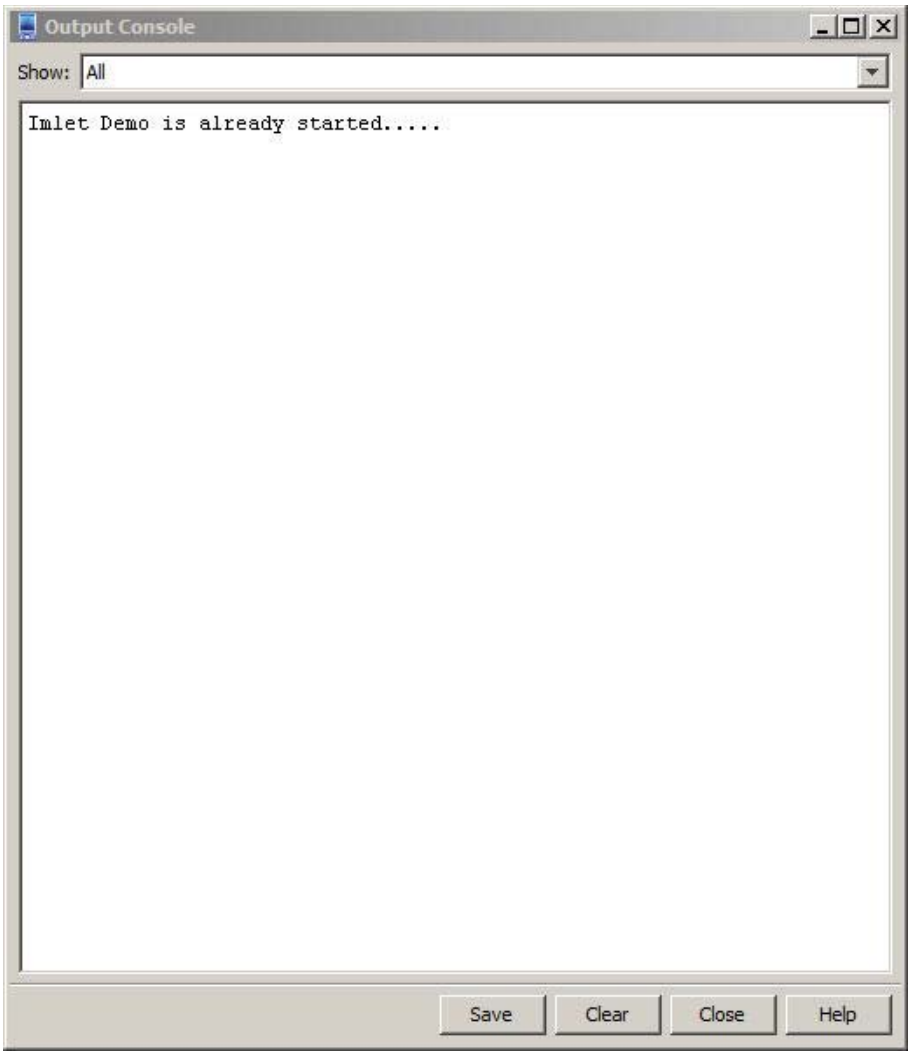
- Install or Reload an application
- Remove an application
- Display Info about an application
- Display Output about an application in a separately-displayed Output Console

Figure 3–3 Buttons on the AMS Tab

The Output Console Window

The Output Console window of the AMS tab allows you to display logging information about a selected application, as shown in Figure 3–4.

Figure 3–4 The AMS Tab Output Console

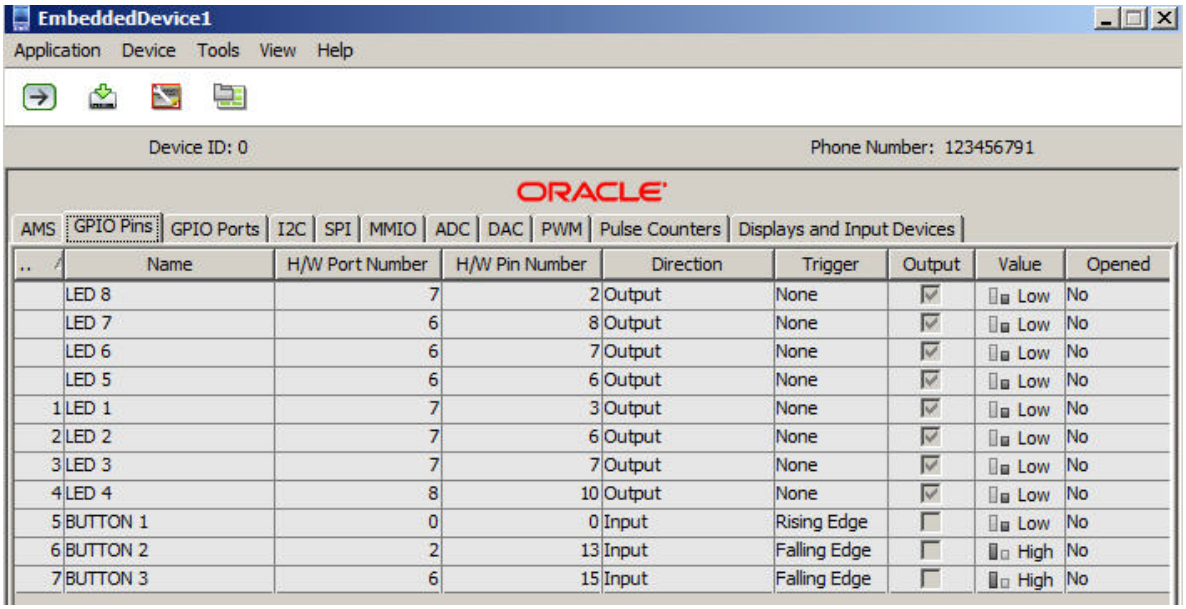


Output Console information can be Error output, Standard output, or All output. The selected output can be saved to a file in your file system by selecting the Save button.

The GPIO Pins Tab

The GPIO tab lists the emulator's current General Purpose I/O (GPIO) pins and their directional state (output). For GPIO pins, the current value is shown as *low* or *high*. The GPIO Pins tab is shown in Figure 3–5.

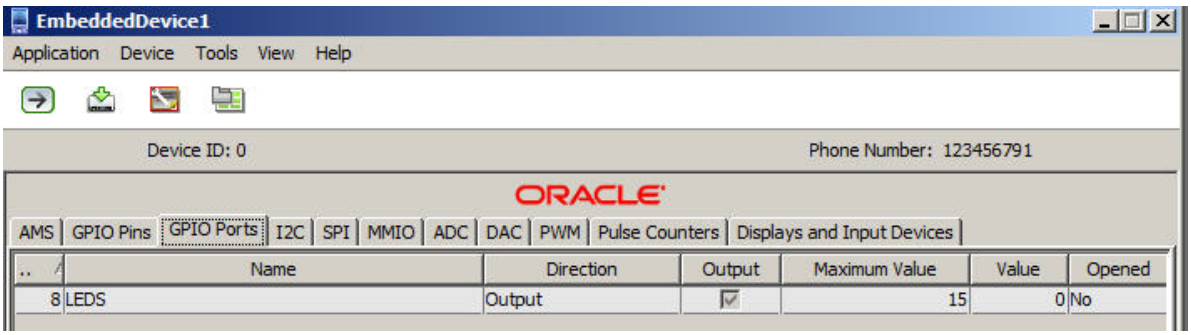
Figure 3–5 The Emulator GPIO Pins Tab



The GPIO Ports Tab

For GPIO ports, the port ID and port Name is shown, along with the port's Maximum Value and current Value. If you select the line that displays the available ports, it displays the pins bound to that specific port, as shown in Figure 3–6.

Figure 3–6 The Emulator GPIO Ports Tab



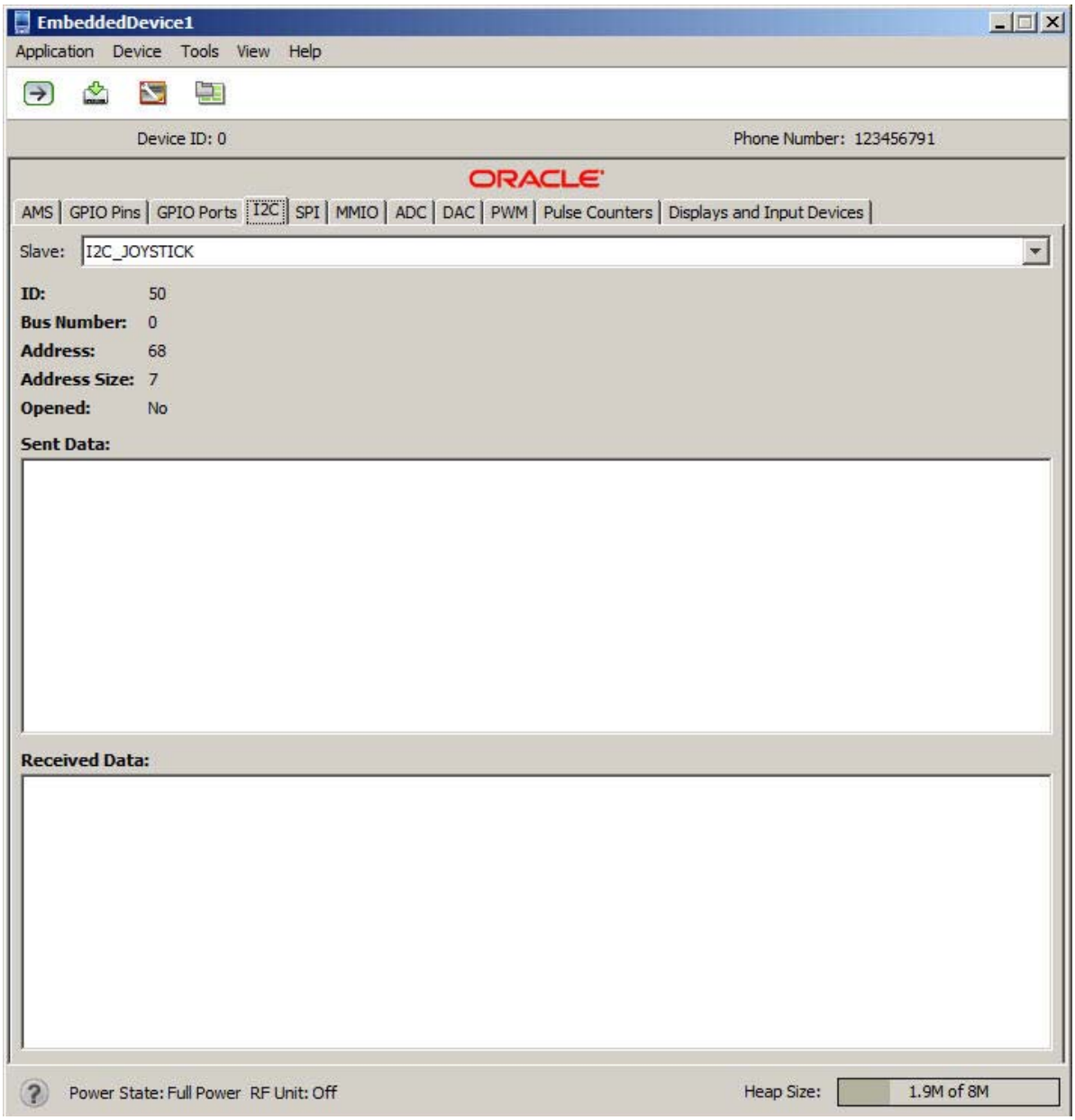
Selecting the line that displays the available ports shows the pins bound to that port, as shown in Figure 3–7.

Figure 3–7 The GPIO Ports Bound Pins

The I2C Tab

The Inter-Integrated Circuit (I2C) tab emulates a simple peripheral slave device that echoes back any data that is sent to it. Both the sent and received data are shown in their appropriate window panes, as shown in Figure 3–8.

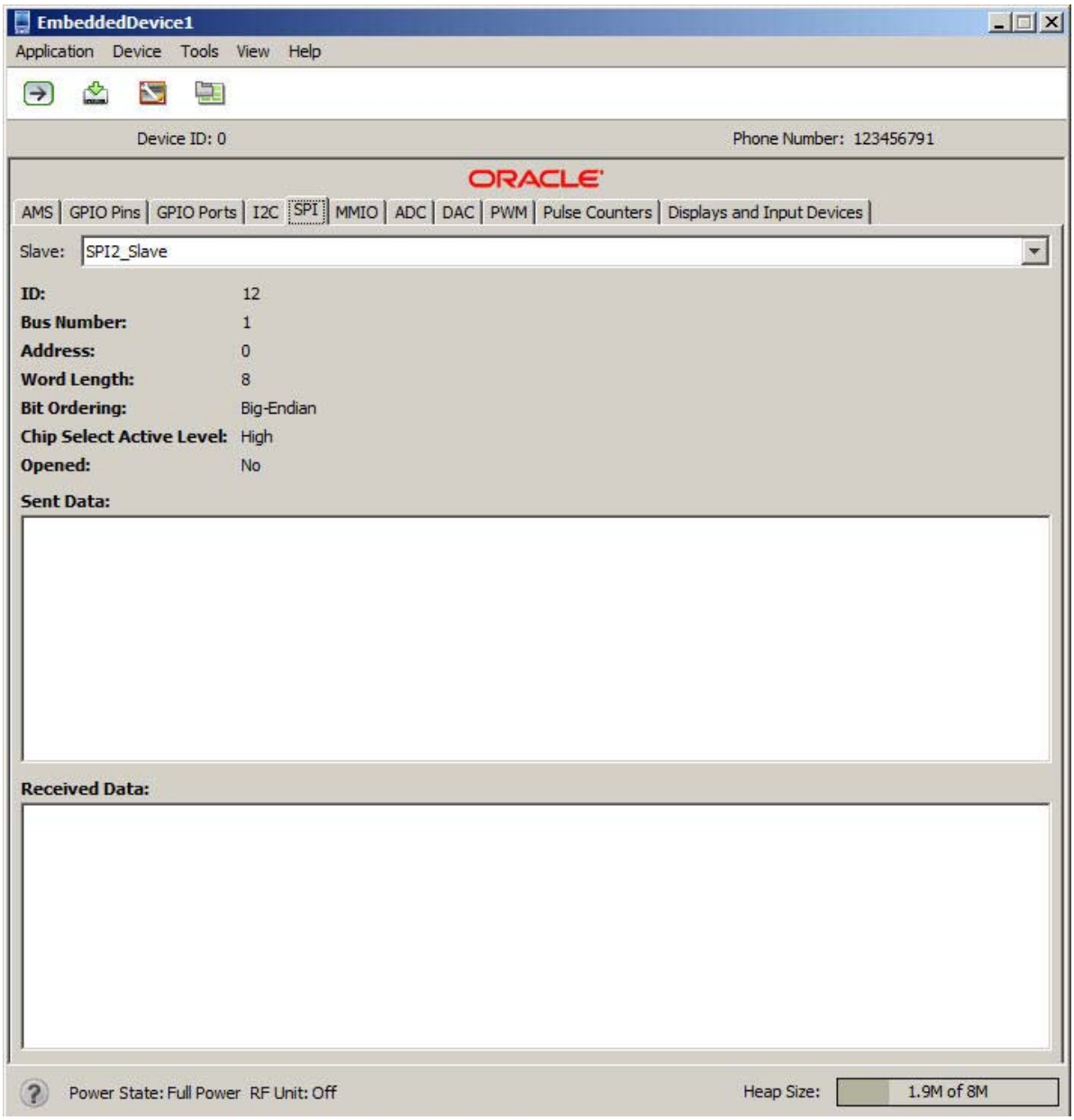
Figure 3–8 The Emulator I2C Tab



The SPI Tab

The Serial Peripheral Interface (SPI) tab is similar to the I2C tab. It emulates a simple peripheral slave device that echoes back any data that is sent to it. Both the sent and received data are shown in their appropriate tabs, as shown in Figure 3–9.

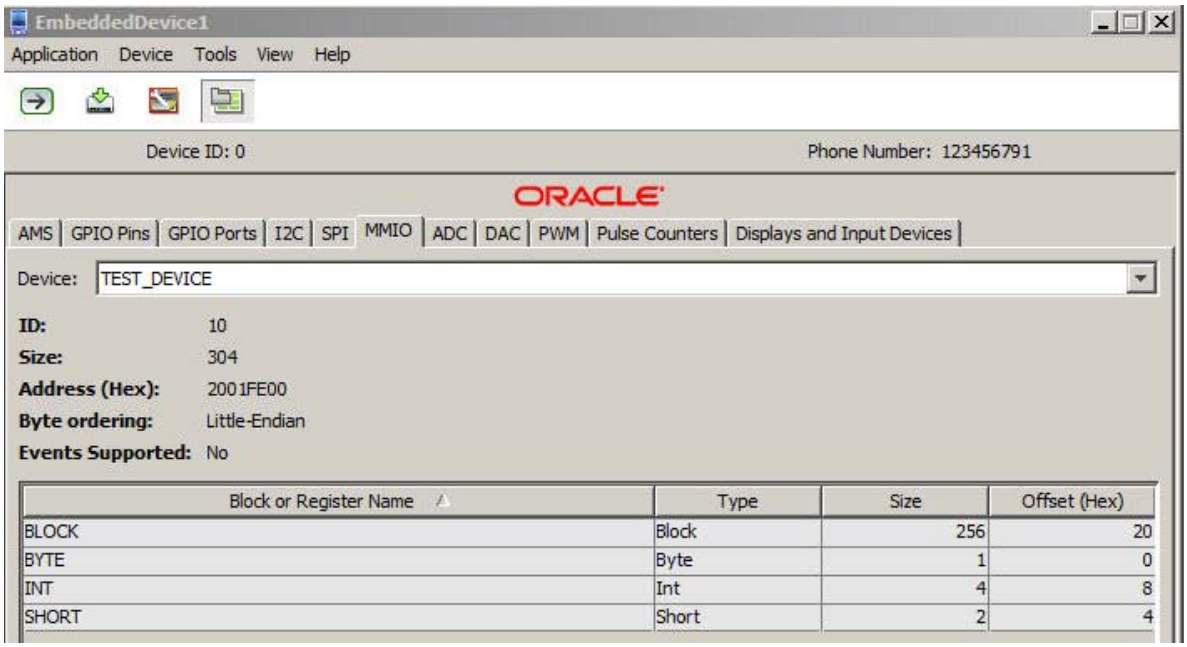
Figure 3–9 The Emulator SPI Tab



The MMIO Tab

The MMIO tab emulates the Memory-Mapped I/O (MMIO) interface bus. It displays an information table about the selected device, as shown in Figure 3–10.

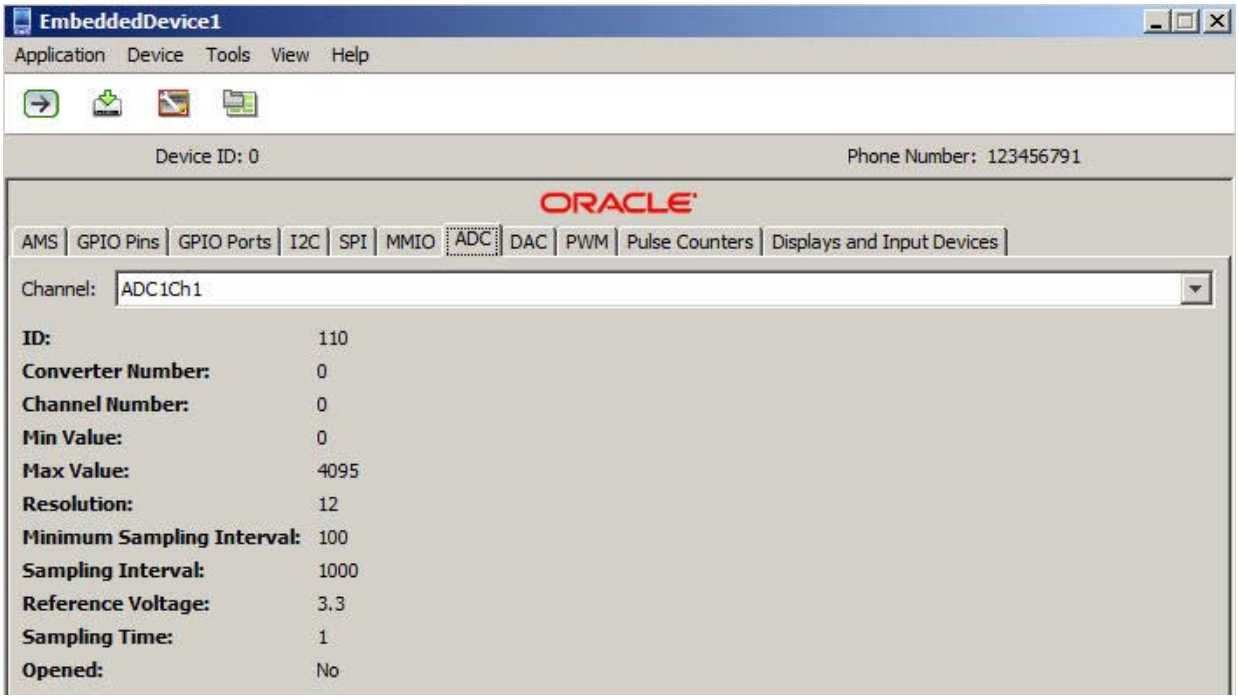
Figure 3–10 The Emulator MMIO Tab



The ADC Tab

The Analog-to-Digital Converter (ADC) tab displays current Channel information, such as ID, Converter Number, Minimum and Maximum Values, Sampling Intervals, and more, as shown in Figure 3–11.

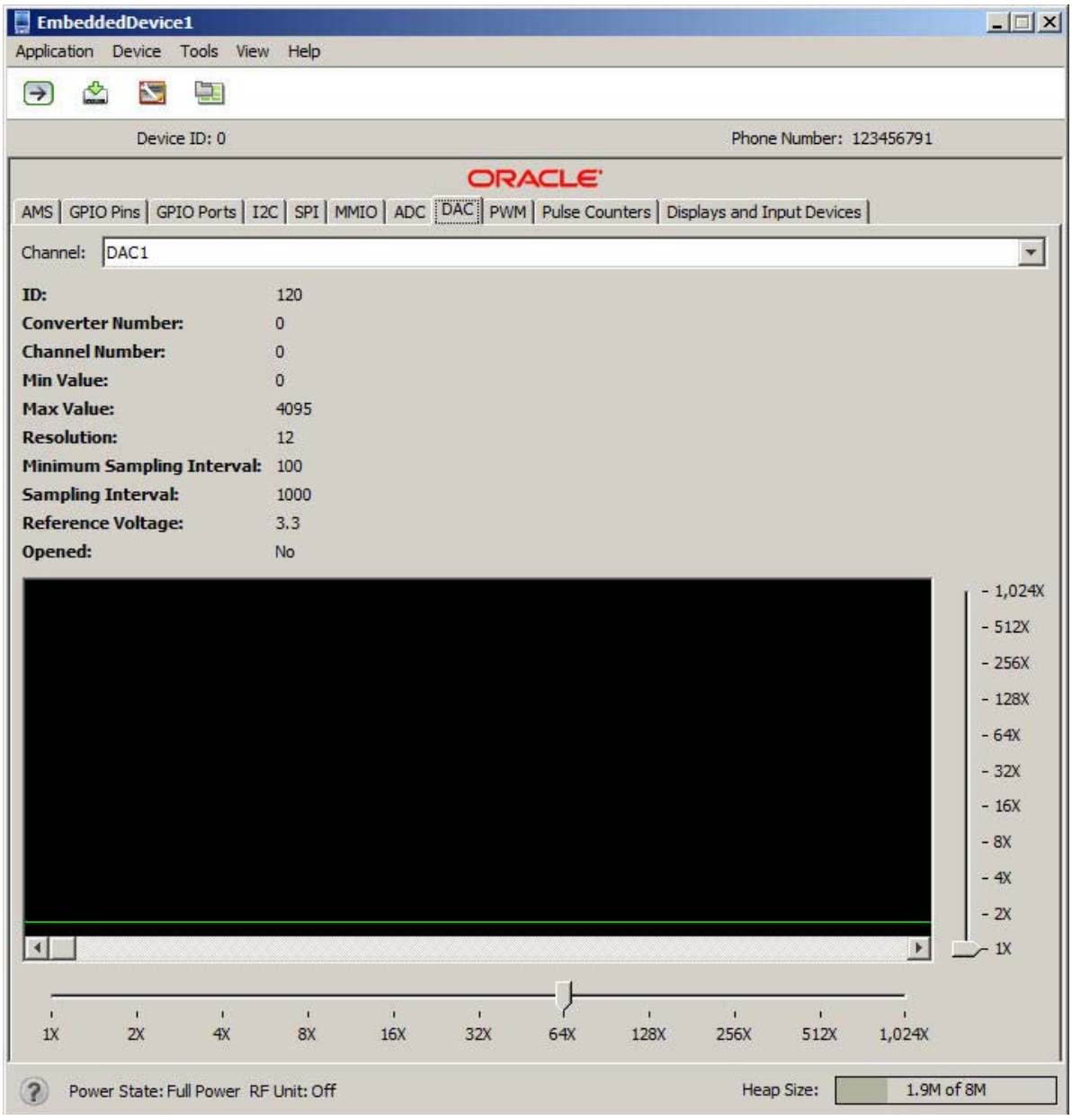
Figure 3–11 The Emulator ADC Tab



The DAC Tab

The Digital-to-Analog Converter (DAC) tab displays current Channel information, such as Minimum and Maximum Values, Sampling Interval, Reference Voltage, and more, as shown in Figure 3–12.

Figure 3–12 The Emulator DAC Tab

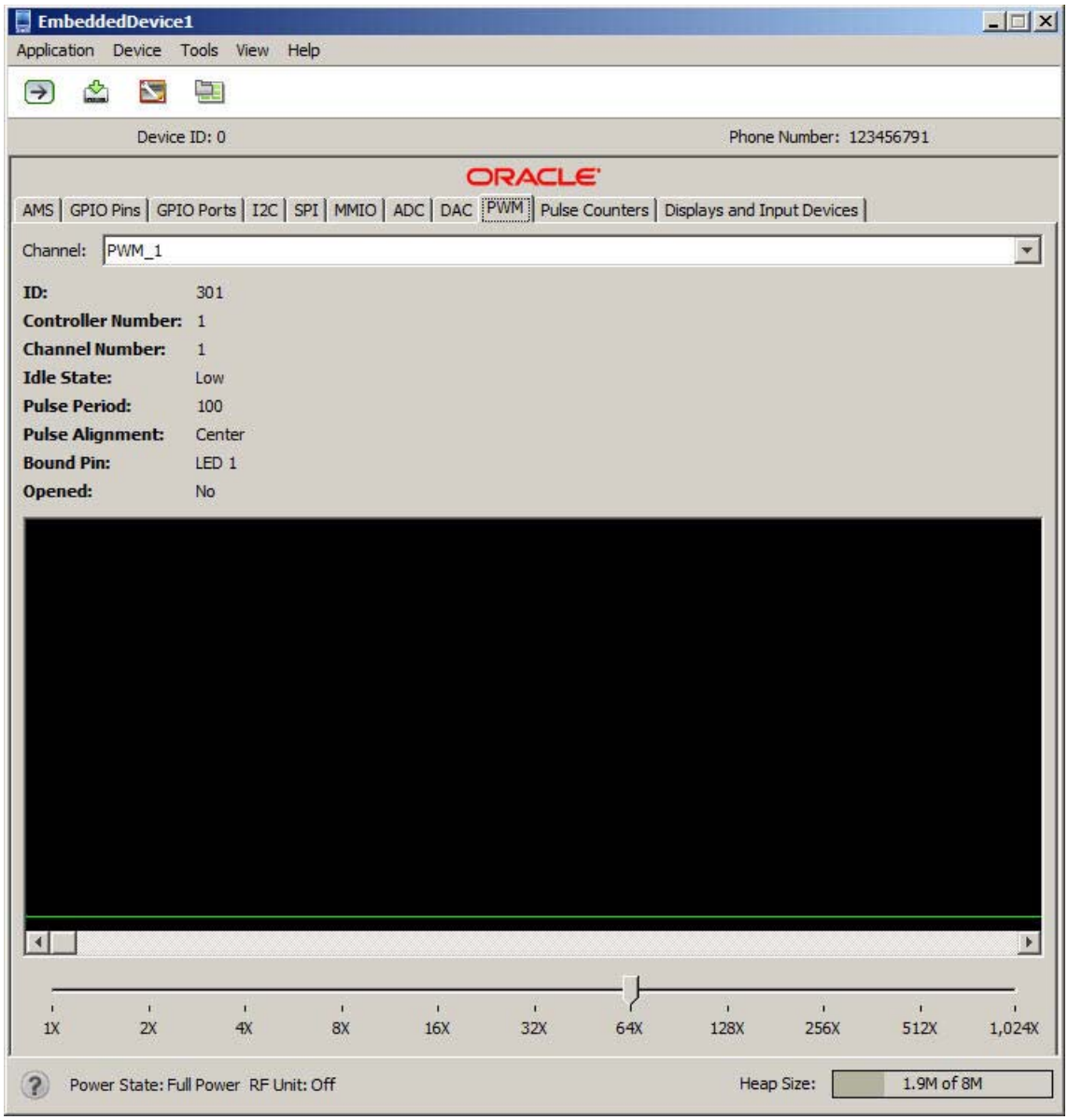


The PWM Tab

The Pulse Width Modulation (PWM) tab allows you to monitor a signal (pulse) that can be encoded for a specific width (duration). It displays information about a specific pulse counter, such as its configuration and current state, as shown in Figure 3–13.

To encode and send a pulse of a specific duration, use the PWM tab of the External Events Generator. For more information, see Chapter 4.

Figure 3–13 The Pulse Width Modulation Tab

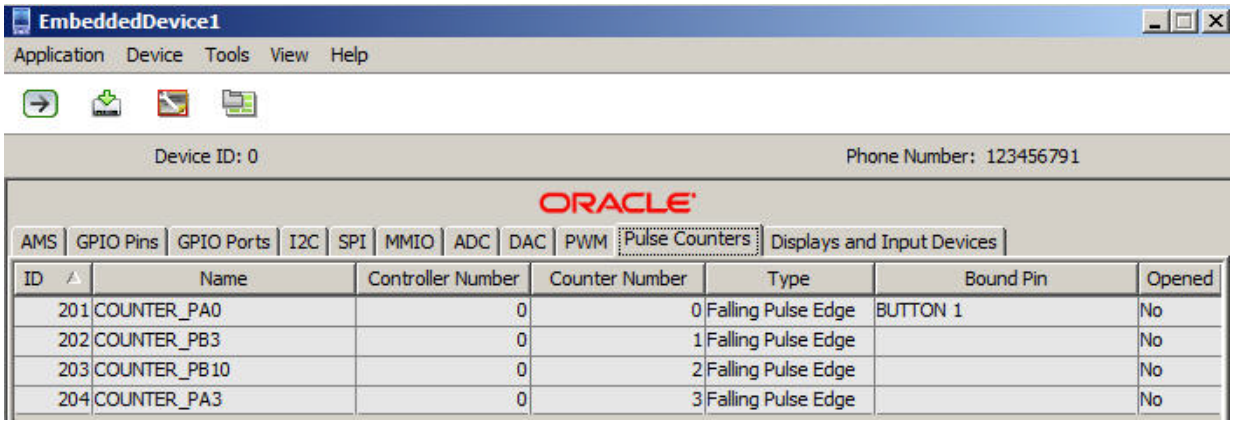


The Pulse Counters Tab

The Pulse Counters tab allows you to send pulse information to as many as four counters. as shown in Figure 3–14. There are four kinds of pulses:

- **Falling Pulse Edge** - Can only be bound to input pins with the Falling Edge trigger.
- **Rising Pulse Edge** - Can only be bound to input pins with the Rising Edge trigger.
- **Negative Pulse Edge** - Can only be bound to input pins with the Both Edges trigger.
- **Positive Pulse Edge** - Can only be bound to input pins with the Both Edges trigger.

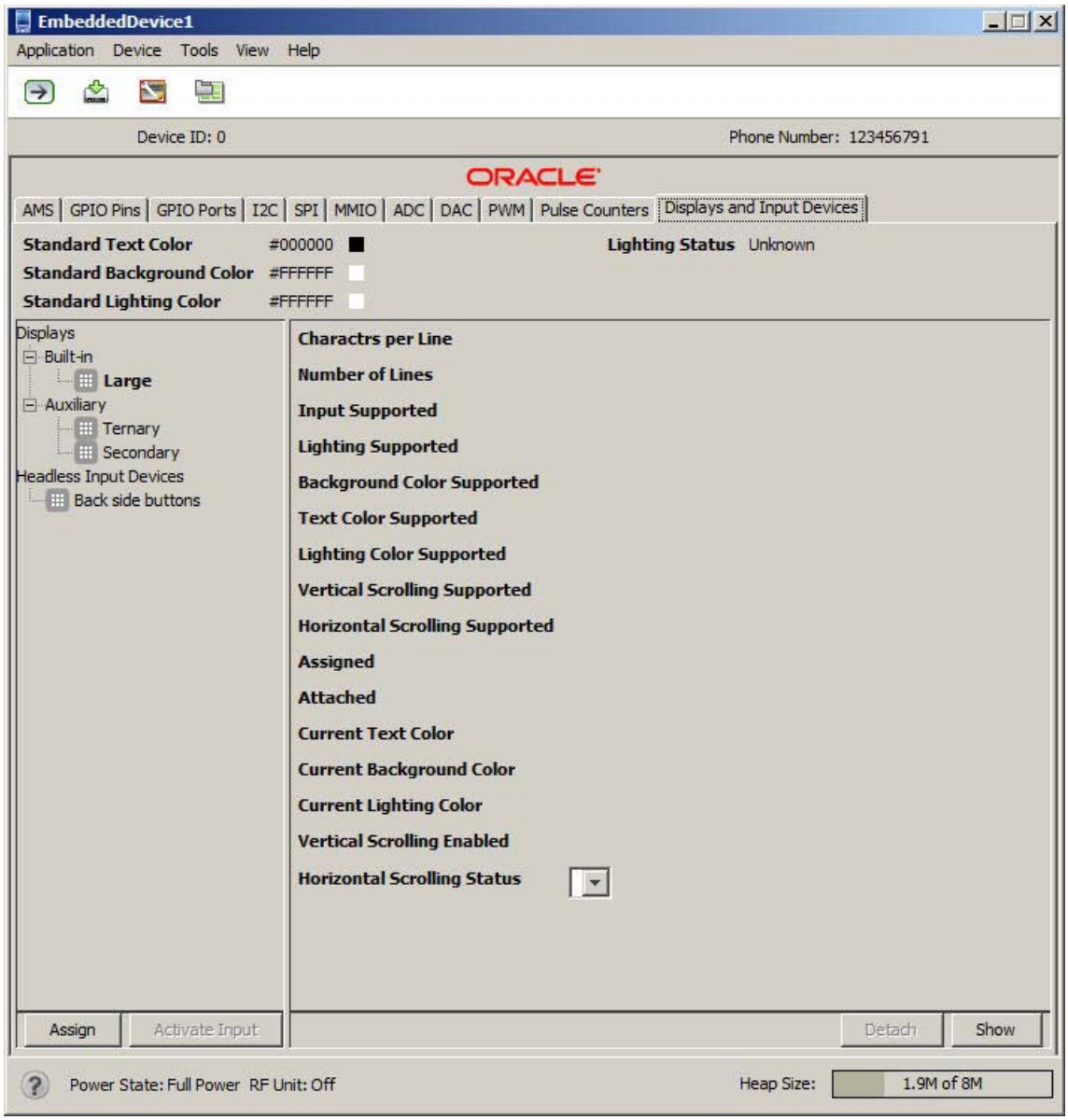
Figure 3–14 The Emulator Pulse Counters Tab



The Displays and Input Devices Tab

The Displays and Input Devices tab allows you to gather information about specific displays attached to your device. This includes both primary and auxiliary displays. This tab also displays information about any input devices, such as a Qualcomm Orion IoE or Raspberry Pi embedded platform, as shown in Figure 3–15.

Figure 3–15 The Displays and Input Devices Tab



Using the Menus

The menus on the Oracle Java ME SDK 8 EA 2 emulator allow you to do a number of operations, including running and installing IMlet suites, managing landmarks, file systems, and access points, and running the External Events Generator.

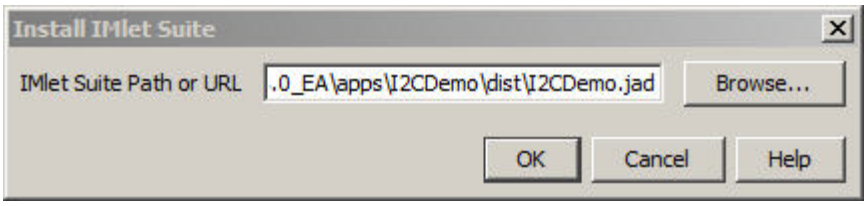
The Application Menu

The Application menu lets you install and run IMlet suites.

- To install an IMlet suite, select **Application > Install IMlet Suite**.

  This displays the Install IMlet Suite box, as shown in Figure 3–16. Click OK.

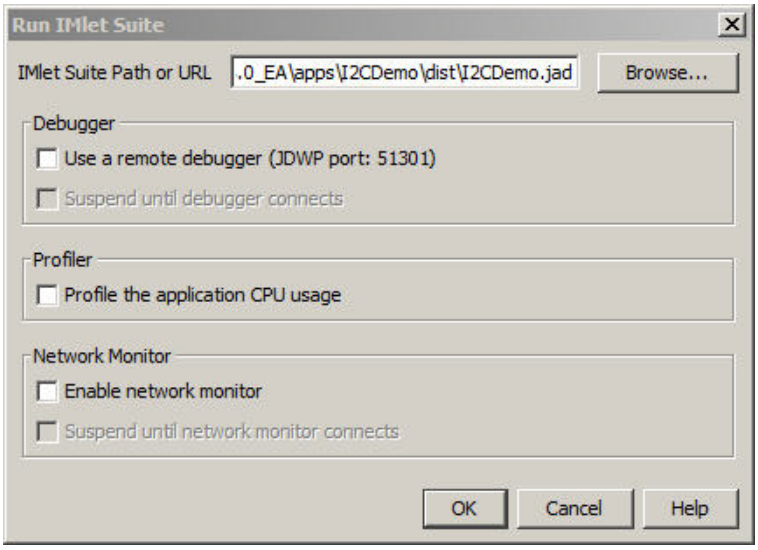
Figure 3–16 The Install IMlet Suite Box



- To run an IMlet suite, select **Application > Run IMlet Suite**.

  This displays the Run IMlet Suite box, as shown in Figure 3–17. Enter the path or URL for the IMlet suite and click OK.

Figure 3–17 The Run IMlet Suite Box



- Exit. Shuts down the emulator process and quits.

The Device Menu

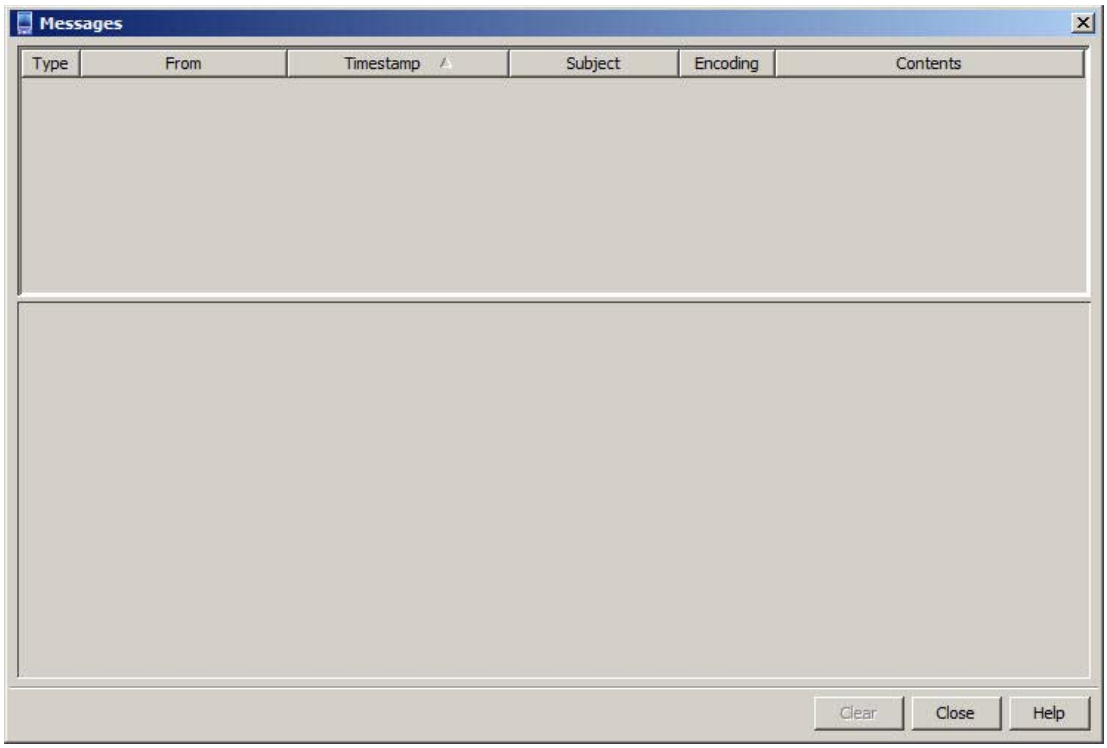
The Device menu allows you to see messages in the Messages box that are directed at the currently-running emulator from other devices. Messages provide information such as the timestamp, subject, encoding, and other information, as shown in Figure 3–18.

Select **Device > Messages** to see what is written in the message area. Messages are sent to a device in the following cases:

- An SMS message is sent without a port in the address (or the port number is 0)
- An SMS text message is sent with a port in the address, but there is not a Java ME application listening on the specified port

To test sending messages to the device use the WMA Console in NetBeans 8.0 Beta, or from the command line, use `wma-tool.exe` to send SMS messages.

Figure 3–18 The View Menu Messages Box



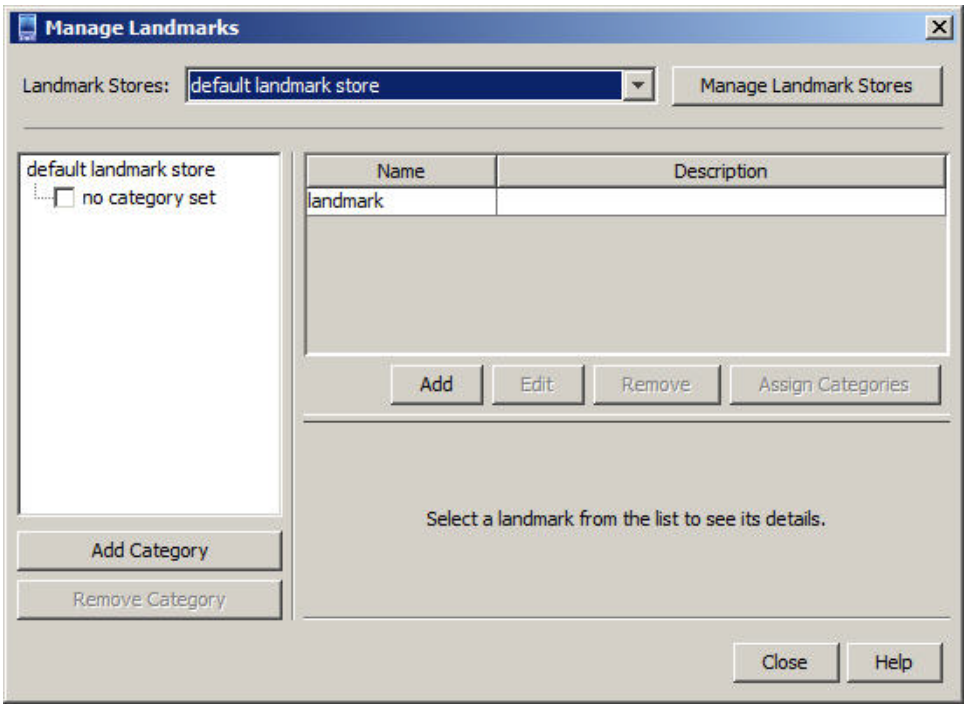
The Tools Menu

The Tools menu allows you to manage landmarks, file systems, and access points. It also provides another way to launch the External Events Generator.

- **Manage Landmarks**. Landmarks are used to emulate location information for an embedded device.

  You can use the default Landmark store, create new landmark stores, define landmark categories, and carry out other landmark operations, as shown in Figure 3–19.

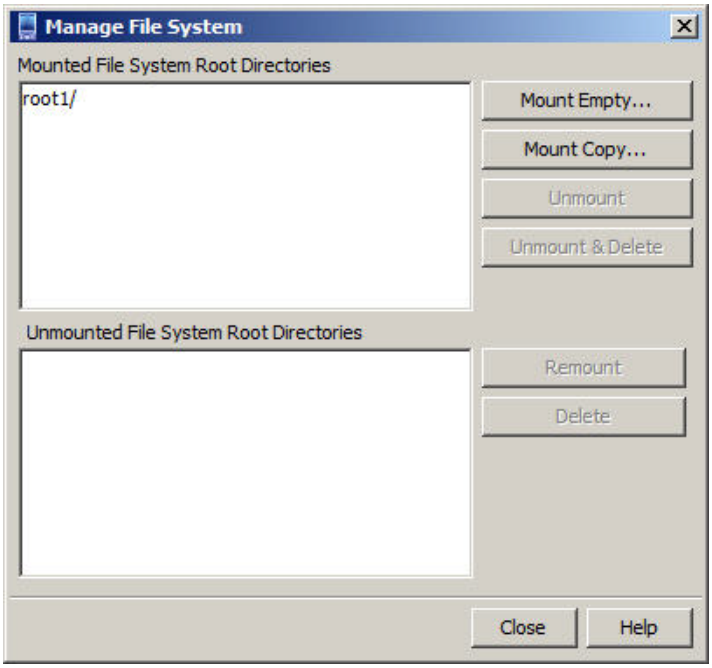
Figure 3–19 The Manage Landmarks Box



- **Manage File System**. Managing a file system allows you to mount a file tree to your emulation environment, providing access to components not installed in your immediate location.

  Once a file tree is mounted, it can also be unmounted, as shown in Figure 3–20.

Figure 3–20 The Manage File System Box

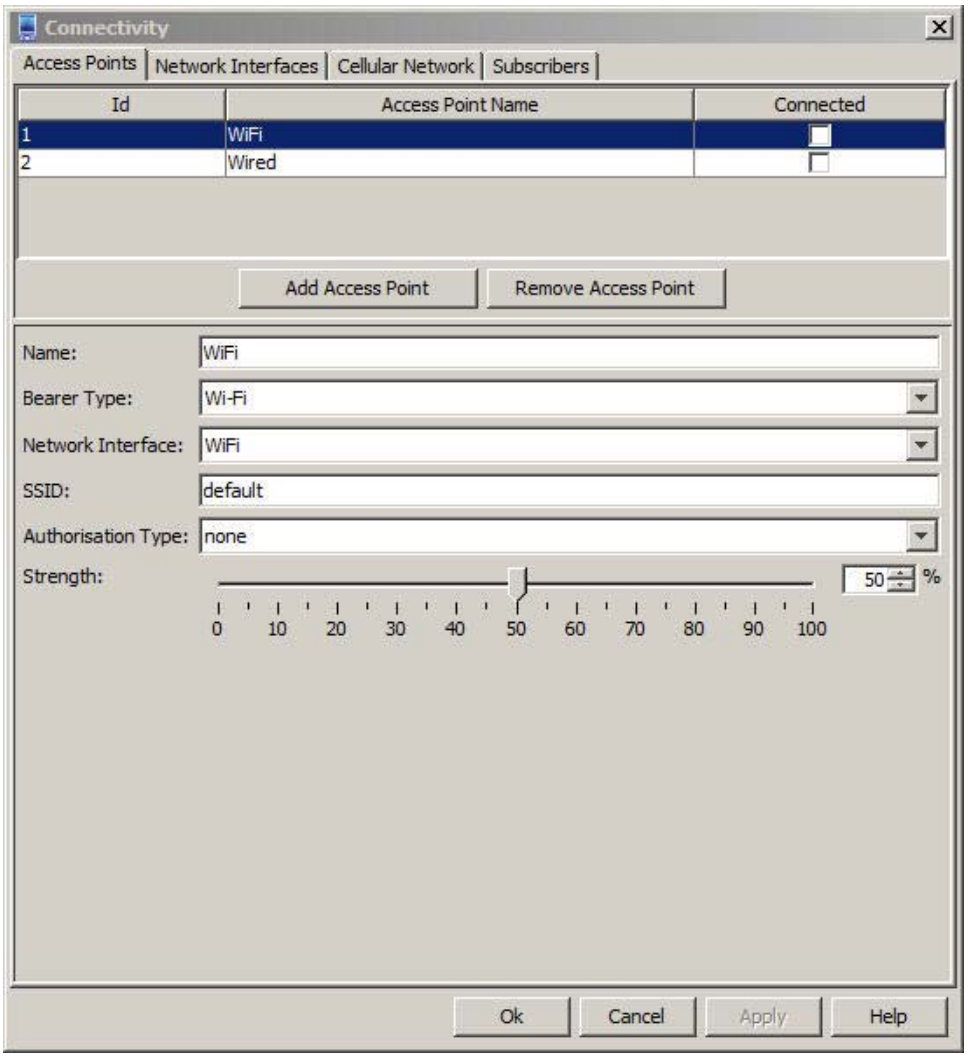


- **Connectivity**. Setting Connectivity allows you to define the network connectivity for your embedded emulation environment.

As shown in Figure 3–21, four tabs are available on the Connectivity screen:

- **Access Points**. Allows you to add and remove access points, and characteristics of a specific access point, such as the bearer type, network interface, and the service set ID (SSID).
- **Network Interfaces**. Allows you to add and remove network interfaces, and define characteristics of a specific network interface, such as the type, timeout interval, and connection interface device.
- **Cellular Network**. Allows you to add and remove a cellular network, and define characteristics for a specific cellular network, such as CDMA or 3GPP, connection protocol, and other details.
- **Subscribers**. Allows you to add or remove a subscriber, and define characteristics for a subscribers, such as phone number, network type, international mobile subscriber identity (IMSI), and other details.

Figure 3–21 The Connectivity Box



- **External Events Generator**. For more information on the External Events Generator, see Chapter 4, "Using the External Events Generator."

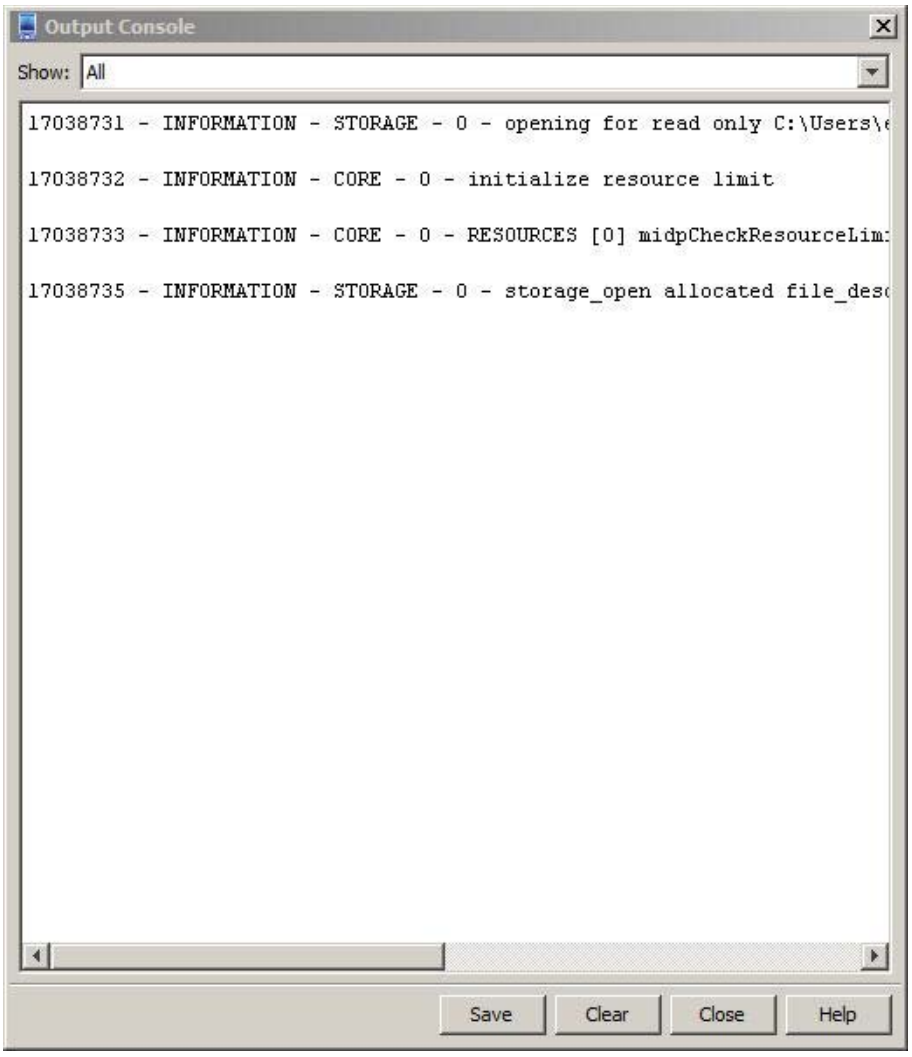
The View Menu

The View menu allows you to view output in a device log or output console, to set your Embedded emulator so it always appears on top on your desktop, and show additional information on exit.

- **Output Console....** Allows you to see output from an application running in the emulator, as shown in Figure 3–22. You have three choices for filtering output: **All**, **Standard Output**, and **Error Output**.

  You can save system output information to a text file by clicking **Save** in the output window.

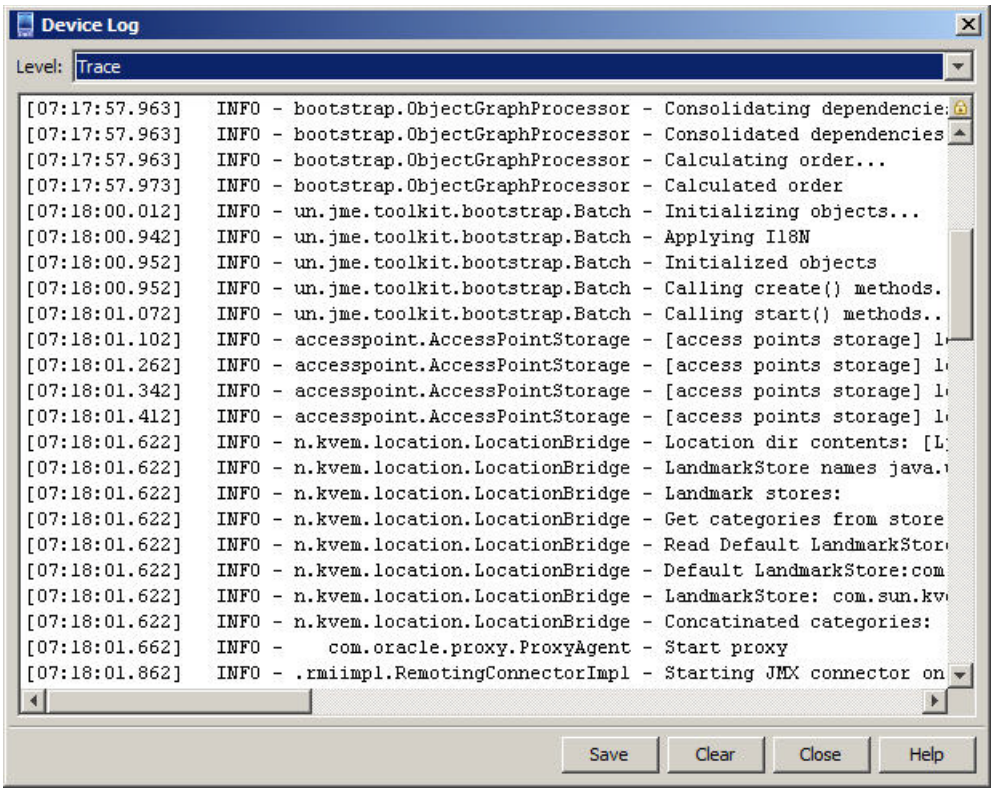
Figure 3–22 The Output Console Window



- **Device Log**. Allows you to display logging info, as shown in Figure 3–23. You have several levels of logging to choose from, including Trace, Debug, info, Warn, Error, and Fatal.

  You can save log information to a text file by clicking **Save** in the output window.

Figure 3–23 The Device Log Window



- **Always on Top**. Allows you to keep the emulator on top of other windows open on your desktop.
- **Show Hot Deployment Hint on Exit**. Upon exit from the emulator, provides suggestions on how to save your work and easily restart the emulator.

The Help Menu

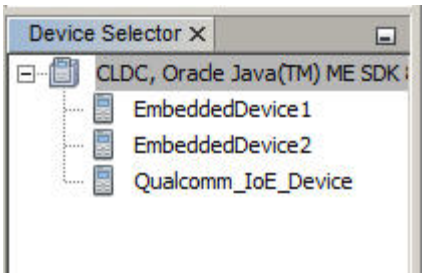
The Help menu allows you to get additional information about the Oracle Java ME SDK 8 EA 2 emulation environment. Put your cursor over any part of the Embedded emulator screen, and select **Help > Help Contents.**

Starting and Running Emulators

The Oracle Java ME SDK 8 EA 2 runs applications on an emulator or an external device.

The Device Connections Manager automatically starts detecting devices when Oracle Java ME SDK 8 EA 2 is installed. The default emulators are automatically found and displayed in the NetBeans 8.0 Beta Device Selector window (**Tools** > **Java ME** > **Device Selector**), as shown in Figure 3–24.

Figure 3–24 Available Devices in the NetBeans 8.0 Beta Device Selector

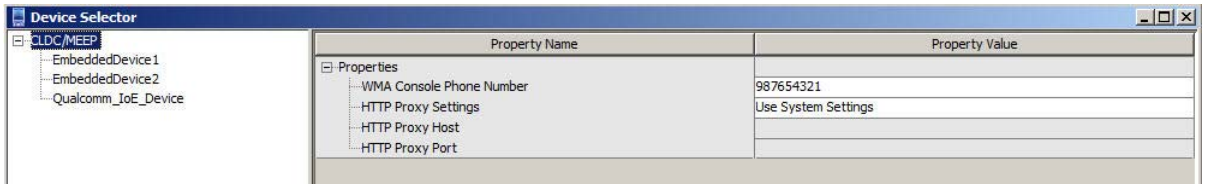


The Device Selector can also be launched from the Windows command line. Enter the following:

C:\>**Java_ME_platform_SDK_8.0_EA\bin\device-selector.exe**

This displays the standalone Device Selector, as shown in Figure 3–25.

Figure 3–25 The Standalone Device Selector



Starting an Emulator

Typically an emulator is launched when a Oracle Java ME SDK 8 EA 2 project is run from the NetBeans 8.0 Beta IDE or from the command line. The default emulator is determined by the Java ME platform selected for the project.

You can open an emulator without running an application from the IDE. From the Windows Start menu, click **Start > All Programs** and select **Java(TM) ME Platform SDK 8.0**. On the pull-right menu, select the default emulator. You can also click the emulator shortcuts installed on your Windows desktop.

To run an application from the emulator, click the **Application** menu and select **Run IMlet Suite**. Provide the path to the application and any other information, and click OK.

The Qualcomm_IoE_Device Emulator

The Qualcomm_I0E_Device emulator is based on MEEP, but for a specific embedded platform, the Qualcomm Orion IoE device. Many of the menus and settings are the same as in the MEEP emulator.

The `Qualcomm_IoE_Device` emulator is launched from the Windows command prompt:

1. Change to the bin directory of the Oracle Java ME SDK 8 EA 2 distribution:

   C:\>**cd Java_ME_platform_SDK_8.0_EA\bin**

2. To start the `Qualcomm_IoE_Device` emulator without a running application, enter:

   C:\>**emulator.exe -Xdevice:Qualcomm_IoE_Device -Xjam**

3. To start the `Qualcomm_IoE_Device` emulator with a running sample application, enter the following command, using the generic command line shown here:

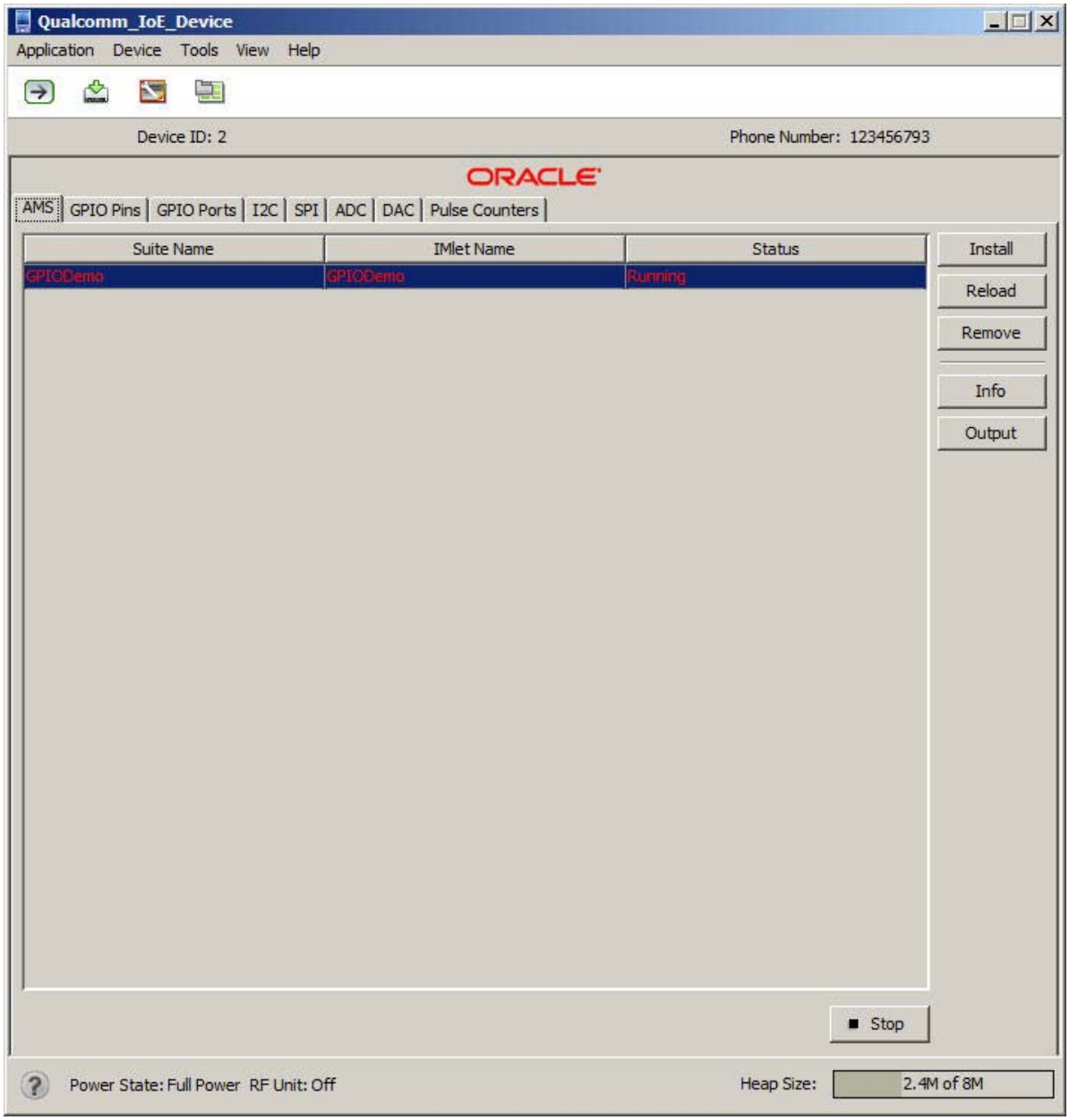
`emulator.exe -Xdevice:Qualcomm_IoE_Device -Xdescriptor:`*location_of_jad_file*

For example, to start the `GPIODemo` sample application, enter the following:

emulator.exe -Xdevice:Qualcomm_IoE_Device -Xdescriptor:C:\Java_ME_platform_SDK_8.0_EA\apps\GPIODemo\GPIODemo.jad

This launches the Qualcomm_IoE_Device emulator, as shown in Figure 3–26.

Figure 3–26 The Qualcomm_IoE_Device Emulator



4

Using the External Events Generator

The Oracle Java ME SDK 8 EA 2 External Events Generator allows you to test the capabilities of your device by simulating events on that device. For example, you can send pulses to a pulse counter.

The External Events Generator is started by clicking on the External Events Generator icon, as shown in Figure 4–1, or by selecting **Tools --> External Events Generator**.

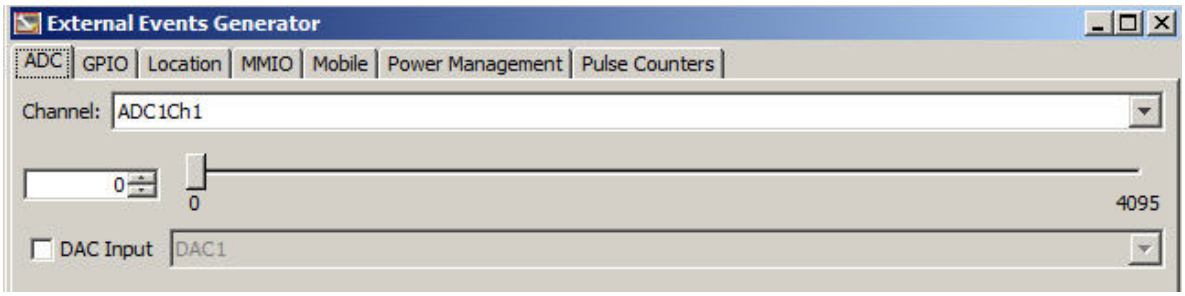
Figure 4–1 External Events Generator Icon

The External Events Generator Screen (EmbeddedDevice1)

Although the External Events Generator functionality is largely the same for both the EmbeddedDevice1 and Qualcomm_IoE_Device emulators, they are not exactly the same. This section describes the tabs and features for the EmbeddedDevice1 External Events Generator.

When the External Events Generator is started the default tab is the Audio-to-Digital Converter (ADC) tab, as shown in Figure 4–2.

Figure 4–2 The External Events Generator



The External Events Generator has the following tabs:

- **ADC**. The Analog-to-Digital Conversion tab allows you to test analog audio input.
- **GPIO**. This General Purpose Input Output (GPIO) option.

  By default, this tab displays ports and pins for a specific device. You can create a custom device to represent a different GPIO device.

See the Device Access API (*installdir*\docs\api\deviceaccess.zip) for a description of the GPIO interface.

- **Location**. Allows you to set and test the location functionality of the device.
- **MMIO**. The memory-mapped I/O (MMIO) option is visible only for the EmbeddedDevice1 emulator. If not already selected, select the following default device:
  - **BIG_ENDIAN_DEVICE.** A Big Endian device that contains all block types: byte, short, int, and block.

Note: If you are using a custom device created with the Custom Device Editor, the MMIO device list might include additional devices. For more information on the Custom Device Editor, see Chapter 5, "Working With Devices"

See the Device Access API *(installdir*\docs\api\deviceaccess.zip) and the Embedded Support API *(installdir*\docs\api\embedded-support-api.zip) for descriptions of the MMIO interface.

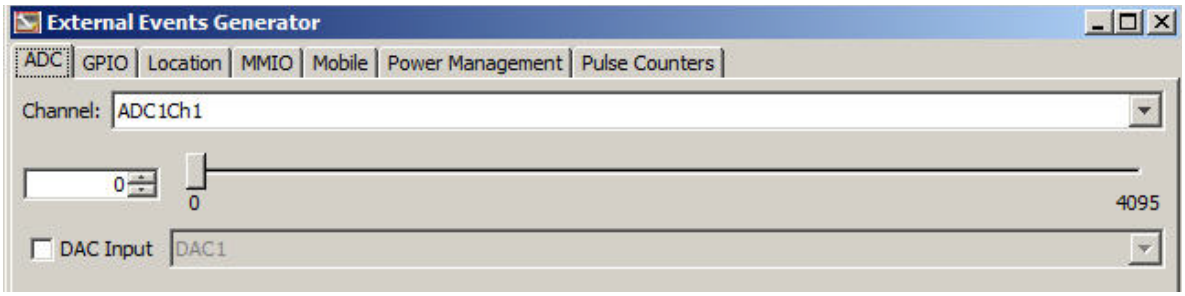
- **Mobile**. Allows you to set the unique International Mobile Subscriber Identity (IMSI) and the International Mobile Station Equipment identity (IMEI) identifiers of the device.
- **Power Management**. This tab allows you to emulate the battery life of an external device, in seconds.
- **Pulse Counter**. This tab displays the current pulse counters on the device. The default configurations are:
  - COUNTER_PA0
  - COUNTER_PB3
  - COUNTER_PB10
  - COUNTER_PA3

  You can configure the pulse counters you want and send a signal to the configured pulse counter by clicking **Send Pulse**.

The ADC Tab

The Analog-to-Digital Converter (ADC) tab allows you to test analog input of ADC channels, as shown in Figure 4–3, including Channel, DAC Input, and minimum and maximum values. You can use the slider to set input voltage manually or connect selected DAC output to selected ADC input.

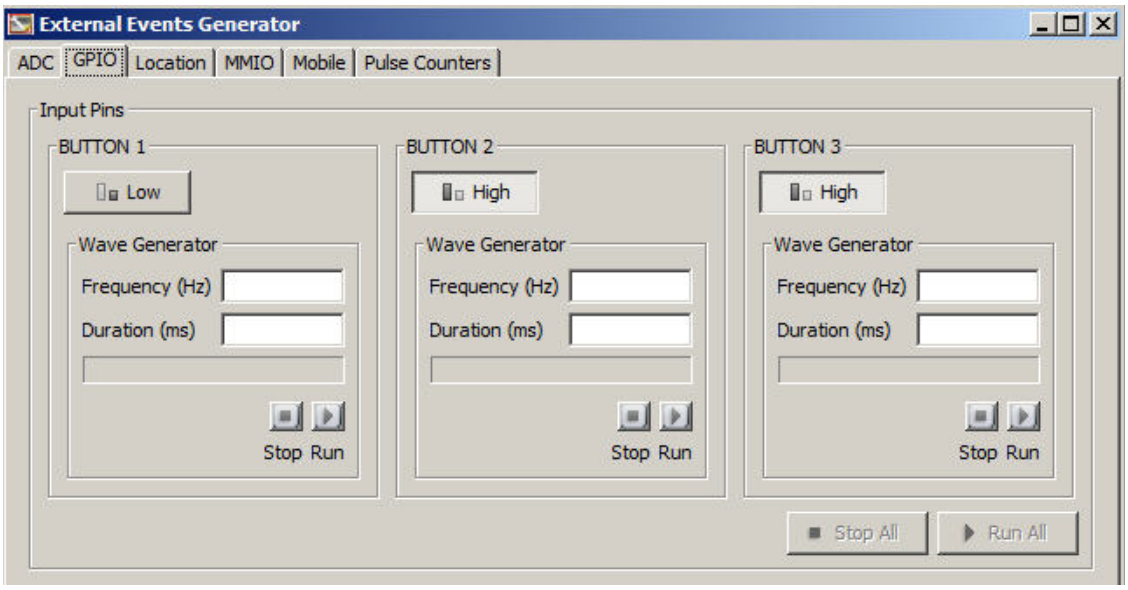
Figure 4–3 The External Events Generator ADC Tab



The GPIO Tab

To generate input events for General Purpose I/O, select the GPIO tab. Here, you can toggle the value of each of the pins from high to low and use a wave generator to simulate a more complex signal to the emulator. The External Events Generator GPIO tab is shown in Figure 4–4.

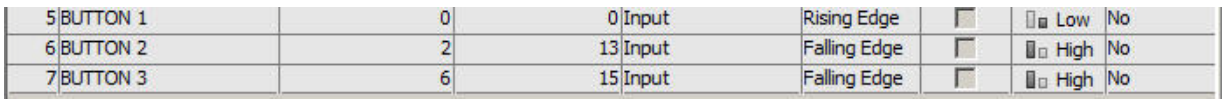
Figure 4–4 The External Events Generator GPIO Tab



To toggle a Button value between Low and High, click on the Button.

When a button value has been changed on the External Events Generator GPIO tab, you see a corresponding change to the Value field of the GPIO Pins tab for the EmbeddedDevice1 emulator, as shown in Figure 4–5.

Figure 4–5 The Button Fields of the EmbeddedDevice1 Emulator GPIO Pins Tab



5	BUTTON 1	0	0	Input	Rising Edge	☐	Low	No
6	BUTTON 2	2	13	Input	Falling Edge	☐	High	No
7	BUTTON 3	6	15	Input	Falling Edge	☐	High	No

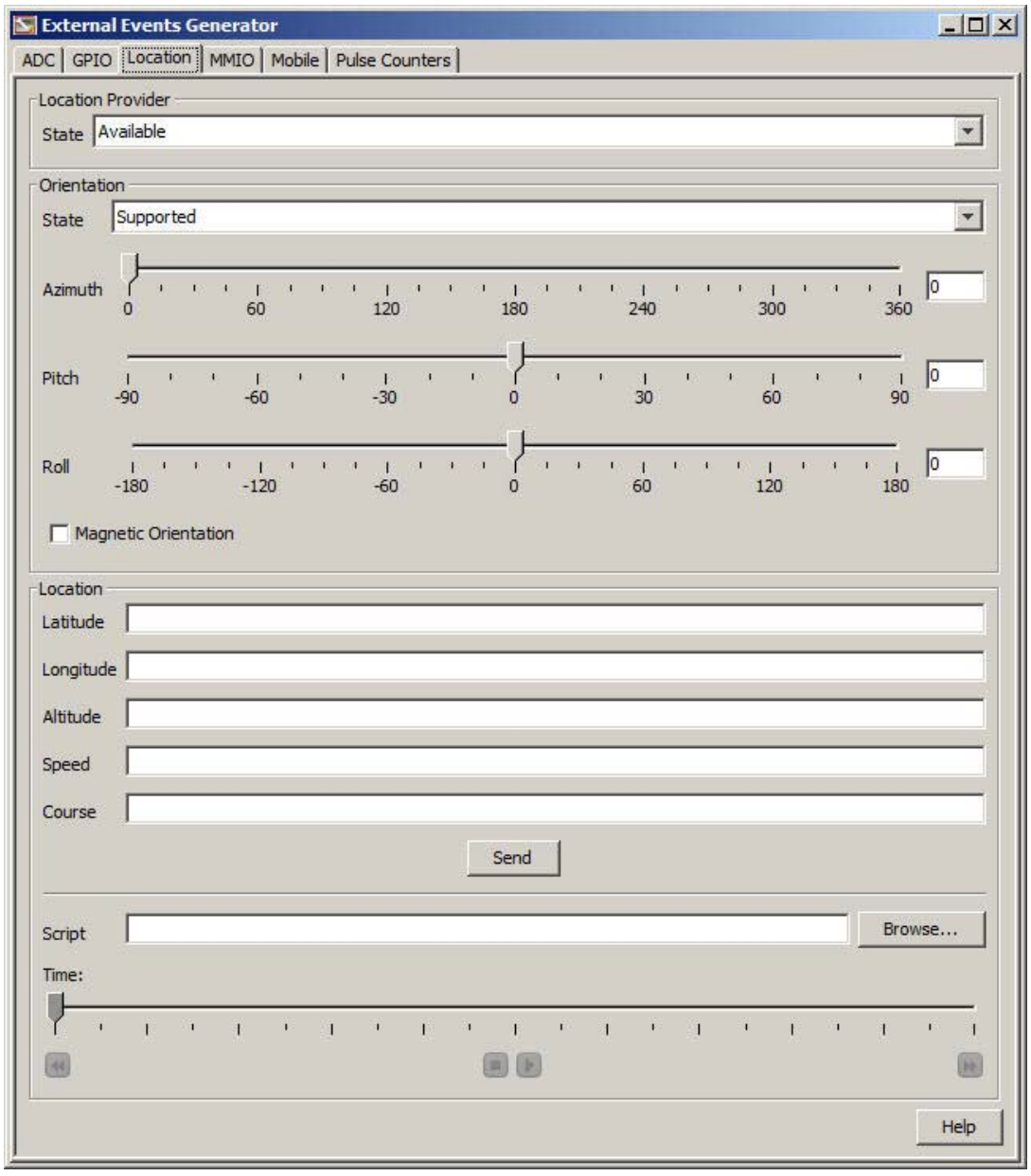
The Location Tab

The Location tab of the External Events Generator allows you to test the Location functionality of your device, as shown in Figure 4–6.

The Location tab has the following fields:

- **Location Provider**. Indicates if the Location Provider is available, temporarily unavailable, or out of service.
- **Orientation**. Indicates if the Orientation is supported, unsupported, or unavailable. Also allows you to set the Azimuth, Pitch, Roll, and Magnetic Orientation of your device.
- **Location**. Allows you to specify specific emulation settings for your device, such as Latitude, Longitude, Altitude, Speed, and Course. It also allows you to browse for and run a specified script or demo program to test the Location capabilities of your device. For a specified script, you can set the duration of the run using the Time slider.

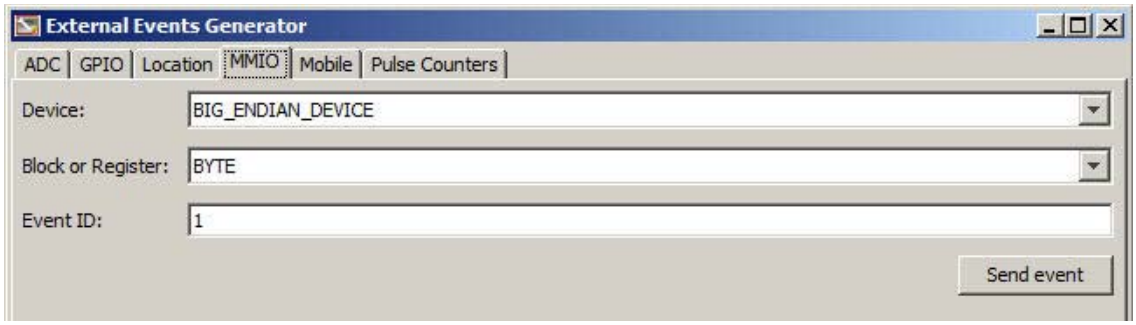
Figure 4–6 The External Events Generator Location Tab



The MMIO Tab

The Memory Mapped I/O tab of the External Events Generator allows the user to simulate sending input event IDs to different MMIO devices that support events. Currently, only the `BIG_ENDIAN_DEVICE` supports events. The MMIO tab is shown in Figure 4–7.

Figure 4–7 The External Events Generator MMIO Tab

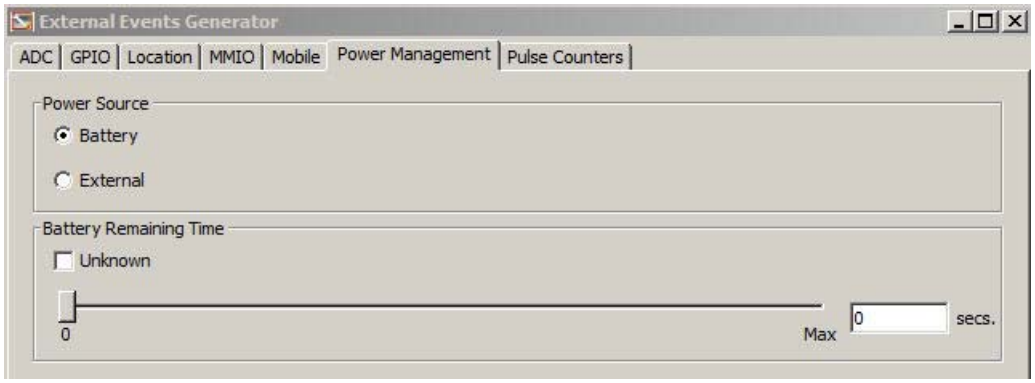


Note: The External Events Generator Mobile tab is not used by embedded applications, so it is not documented here.

The Power Management Tab

The Power Management tab allows you to define power settings for a device battery or external device. The Power Management tab is shown in Figure 4–8.

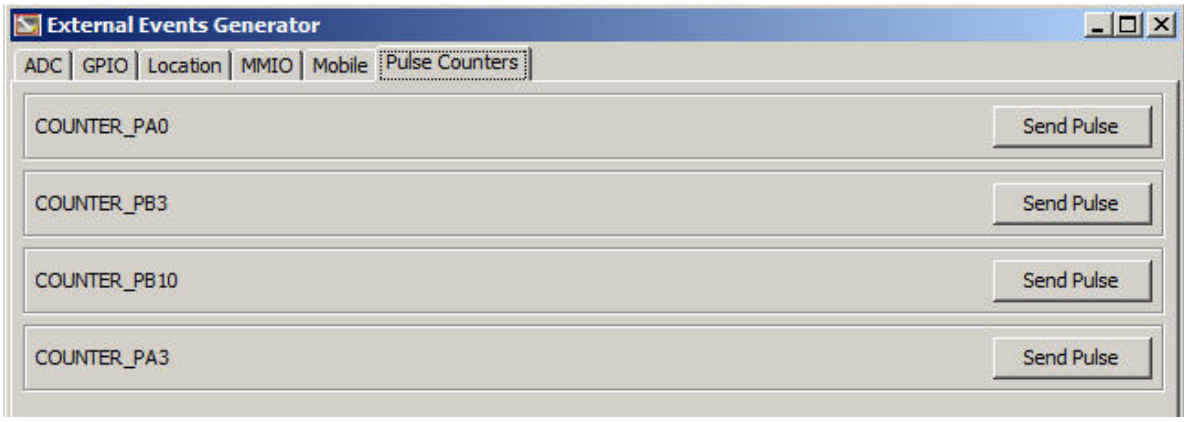
Figure 4–8 The Power Management Tab



The Pulse Counter Tab

The Pulse Counter tab allows you to send a sequence of pulses to the pulse counters available in the current embedded devices, as shown in Figure 4–9.

Figure 4–9 The External Events Generator Pulse Counters Tab



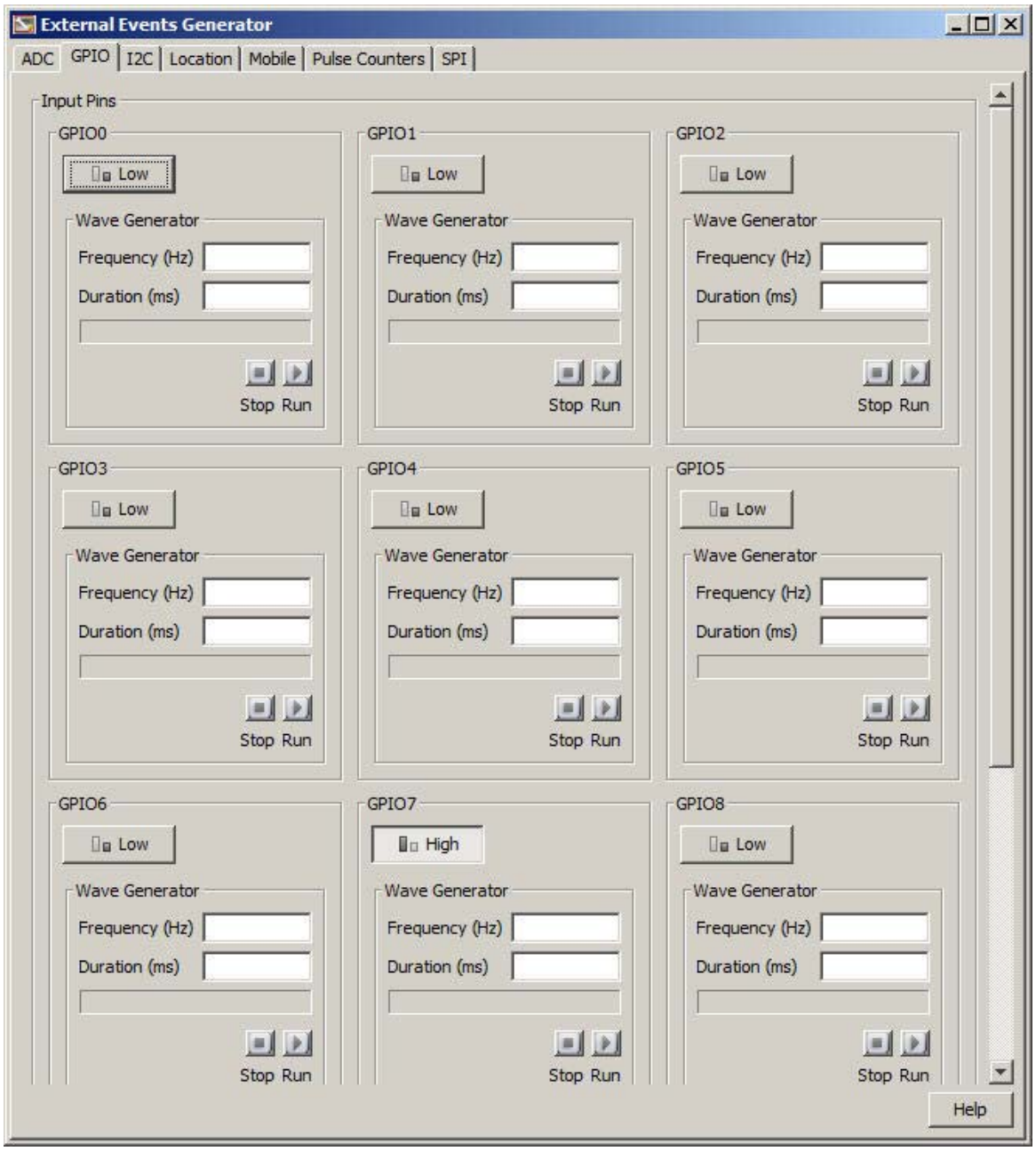
The External Events Generator (Qualcomm_IoE_Device)

The tabs on the External Events Generator for the Qualcomm_IoE_Device emulator largely match those of the EmbeddedDevice1 emulator. Those that differ are shown below.

The GPIO Tab

The GPIO tab for the Qualcomm_IoE_Device External Events Generator is similar in function to the GPIO tab in the EmbeddedDevice1 emulator, but has many more buttons available, as shown in Figure 4–10.

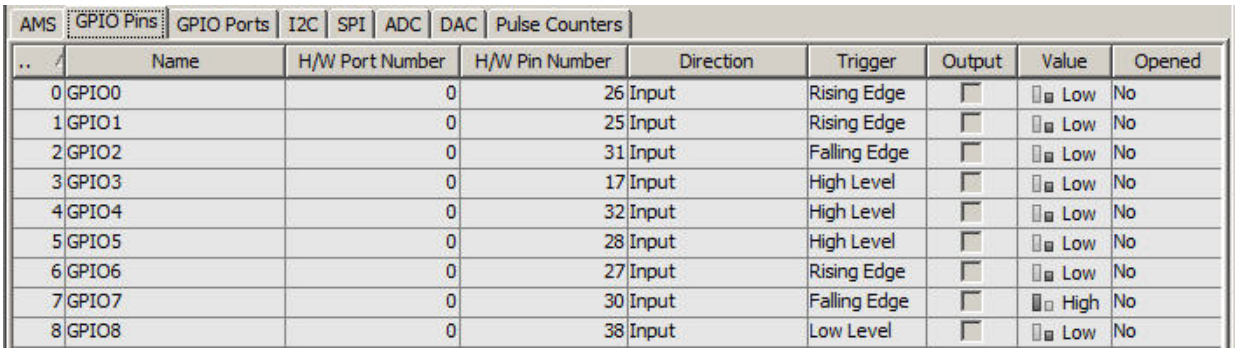
Figure 4–10 The Qualcomm_IoE_Device External Events Generator GPIO Tab



To toggle a Button value between Low and High, click on the Button.

When a button value has been changed on the External Events Generator GPIO tab, you see a corresponding change to the Value field of the GPIO Pins tab for the Qualcomm_IoE_Device emulator, as shown in Figure 4–11.

Figure 4–11 The GPIO Pins Tab of the Qualcomm_IoE_Device Emulator



..	Name	H/W Port Number	H/W Pin Number	Direction	Trigger	Output	Value	Opened
0	GPIO0	0	26	Input	Rising Edge	☐	Low	No
1	GPIO1	0	25	Input	Rising Edge	☐	Low	No
2	GPIO2	0	31	Input	Falling Edge	☐	Low	No
3	GPIO3	0	17	Input	High Level	☐	Low	No
4	GPIO4	0	32	Input	High Level	☐	Low	No
5	GPIO5	0	28	Input	High Level	☐	Low	No
6	GPIO6	0	27	Input	Rising Edge	☐	Low	No
7	GPIO7	0	30	Input	Falling Edge	☐	High	No
8	GPIO8	0	38	Input	Low Level	☐	Low	No

The Inter-Integrated Circuit (I2C) Tab

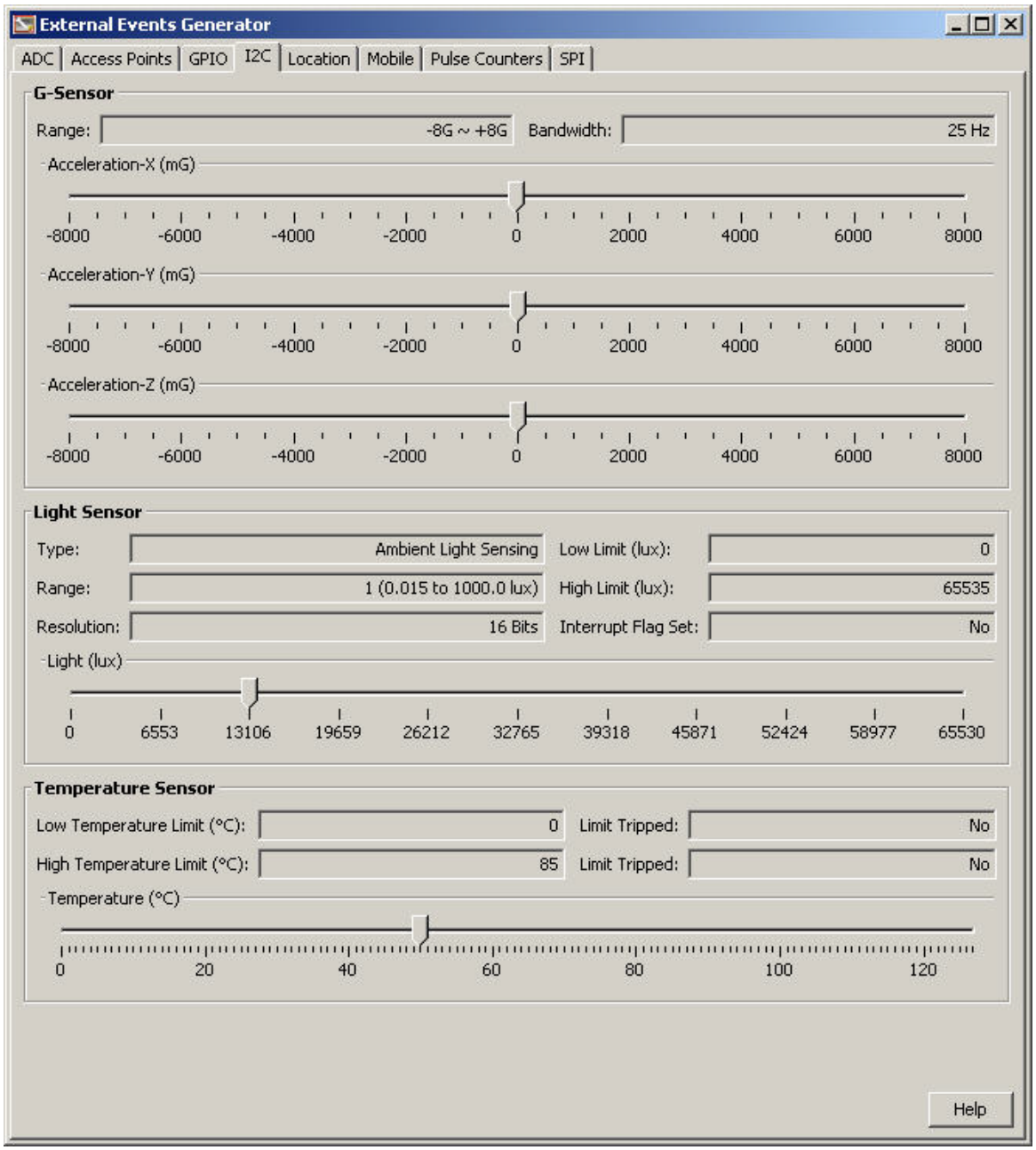
The Inter-Integrated Circuit (I2C) tab displays information for a selected slave device, and data sent to and received from a master device.

The three sample sensors in the I2C tab are:

- **G-Sensor**. Operates in the same way as described in the External Events Generator SPI tab. However, the sample IMLet reads the accelerometer values through the I2C interface. For more information, see "The Serial Peripheral Interface (SPI) Tab."
- **Light Sensor**. Allows the user to test three kinds of light sensing: ambient light, proximity, and Infrared (IR). A lux value can be specified, and the values changed within the lux value range by moving the Light slider.
- **Temperature Sensor**. High and Low temperature values can be set, and temperature values read when the slider is moved.

The Qualcomm_IoE_Device emulator External Events Generator I2C tab is shown in Figure 4–12.

Figure 4–12 The Qualcomm_IoE_Device External Events Generator I2C Tab



The Light Sensor

The Light Sensor has the following fields:

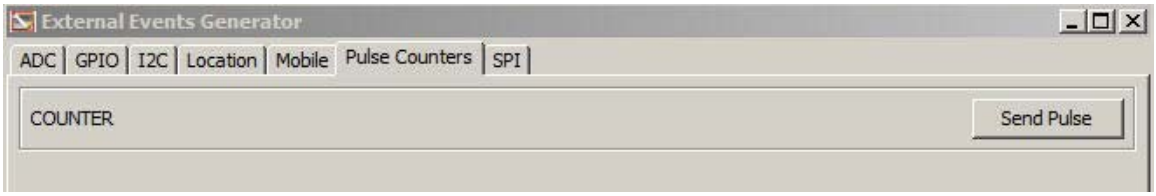
- **Type**. Ambient Light. Allows changes in light brightness to be registered by the emulated sensor.
- **Low Limit (lux)**. The low limit value is `0` lux.
- **Range**. The range is `1` (defined as `0.015` lux to `1000.0` lux)
- **High Limit (lux)**. The high limit value is `65535` lux.
- **Resolution**. Allows `16` bits.

- **Interrupt Flag Set**. When sensor values stay within the defined range, the value of the Interrupt Flag is `No`.
- **Light (lux)**. Allows the user to define the lux value for the application by moving the slider. The range values allowed are `0` lux at the low end and `65535` lux at the high end.

The Pulse Counters Tab

The Pulse Counters tab of the Qualcomm_IoE_Device External Events Generator works the same as the tab in the External Events Generator for the EmbeddedDevice1. However, in the Qualcomm_IoE_Device tab there is only one counter, as shown in Figure 4–13.

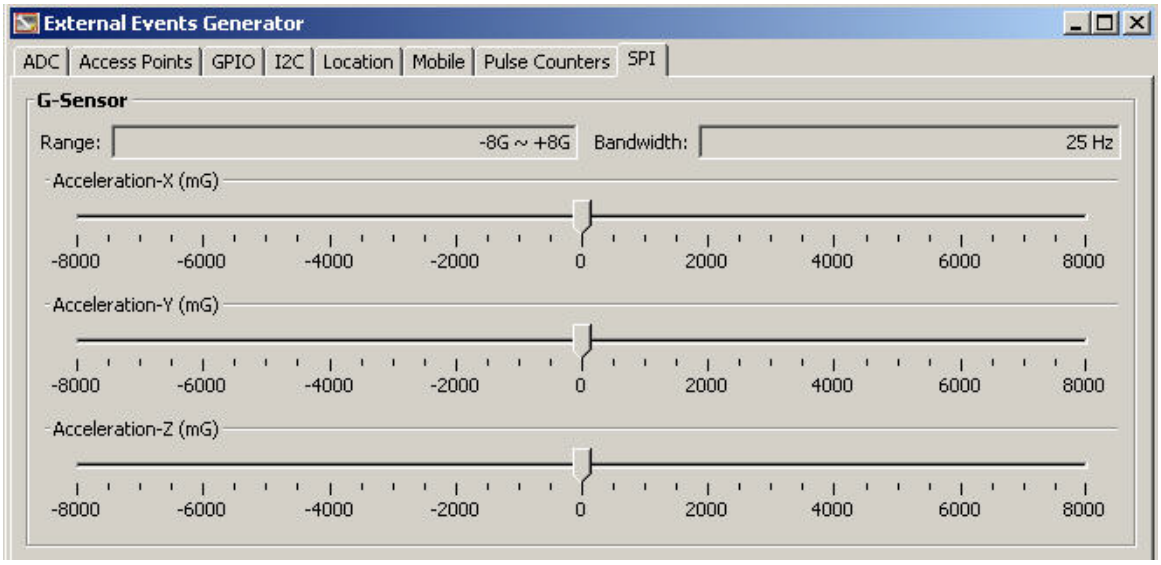
Figure 4–13 The Qualcomm_IoE_Device External Events Generator Pulse Counters Tab



The Serial Peripheral Interface (SPI) Tab

The Qualcomm_IoE_Device emulator External Events Generator SPI tab provides a sample accelerometer sensor, G-Sensor, as shown in Figure 4–14.

Figure 4–14 The Qualcomm_IoE_Device External Events Generator SPI Tab



The G-Sensor sample allows an IMlet to read accelerometer values from the emulator through the Serial Peripheral Interface. Move the slider of the X, Y, and Z acceleration scale to change the x, y, and z values transmitted over SPI. The minimum and maximum values for the sliders in the G-Sensor are defined by the Java ME application.

The G-Sensor sample duplicates in emulation the functionality of the digital, triaxial acceleration sensor on the Qualcomm Orion IoE embedded device. The acceleration

sensor is used to sense tilt, motion, and shock vibration in embedded devices, such as medical instruments, computer peripherals, and monitoring devices.

The G-Sensor Fields

The G-Sensor has the following fields:

- **Range**. Allows between `-8G` and `+8G`.
- **Bandwidth**. Allows `25` Hz.
- **Accelerator-X**. Allows you to select a value within a specified range by moving the slider position.
- **Accelerator-Y**. Allows you to select a value within a specified range by moving the slider position.
- **Accelerator-Z**. Allows you to select a value within a specified range by moving the slider position.

For more information on working with the Qualcomm Orion IoE embedded device, see the Oracle Java ME Embedded *Getting Started Guide for the Reference Platform (Qualcomm Orion IoE)*.

5

Working With Devices

This chapter describes the Oracle Java ME SDK 8 EA 2 components that allow you to work effectively with devices, including making device connections, creating, configuring, and customizing devices, and detecting external embedded devices.

The Oracle Java ME SDK 8 EA 2 provides three components for working with emulated and external devices in the NetBeans 8.0 Beta IDE:

- Device Selector
- Device Connections Manager
- Custom Device Editor

The Device Connections Manager is started automatically when Oracle Java ME SDK 8 EA 2 is installed. However, its primary usefulness is for external embedded hardware is attached to the Windows 7 PC running

The Device Manager

The Device Manager icon is put on your desktop when you first install Oracle Java ME SDK 8 EA 2, as shown in Figure 5–1. It's located in your Windows 7 system tray and can be run without starting the NetBeans 8.0 Beta.

Figure 5–1 The Device Connections Manager Icon

The Device Manager is an Oracle Java ME SDK 8 EA 2 service used to manage both emulators and external devices. The Device Manager automatically detects available devices (including any connected hardware devices) and lists those devices in the Device Selector window.

To use the Device Manager:

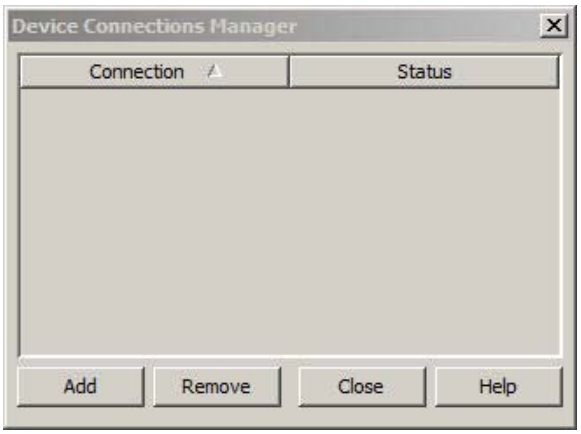
1. If not already running, the Device Manager can be started in two ways:
   - From NetBeans 8.0 Beta, select **Tools** > **Java ME** > **Active Device Manager** > **Oracle Java(TM) ME SDK 8.0 EA**
   - From the Windows command line. Type:

     C:\>**Java_ME_platform_SDK_8.0_EA\bin\device-manager.exe**
2. When the Device Manager is in the Windows system tray, right-click the icon. This displays the Device Connections Manager menu, as shown in Figure 5–2.

Figure 5–2 The Device Connections Manager Menu

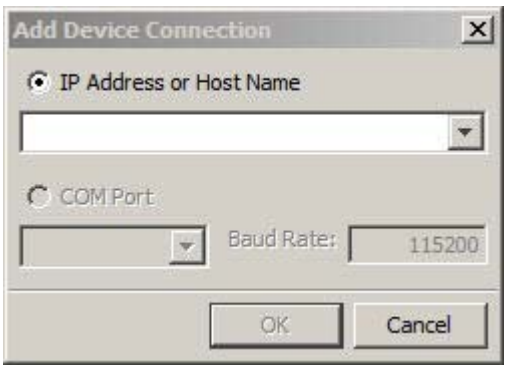
3. Select **Manage Device Connections** to display the Device Connections Manager screen, as shown in Figure 5–3.

Figure 5–3 The Device Connections Manager Screen



4. To add a new device connection, click **Add**. This displays the Add Device Connection box, as shown in Figure 5–4.

Figure 5–4 The Add Device Connection Form



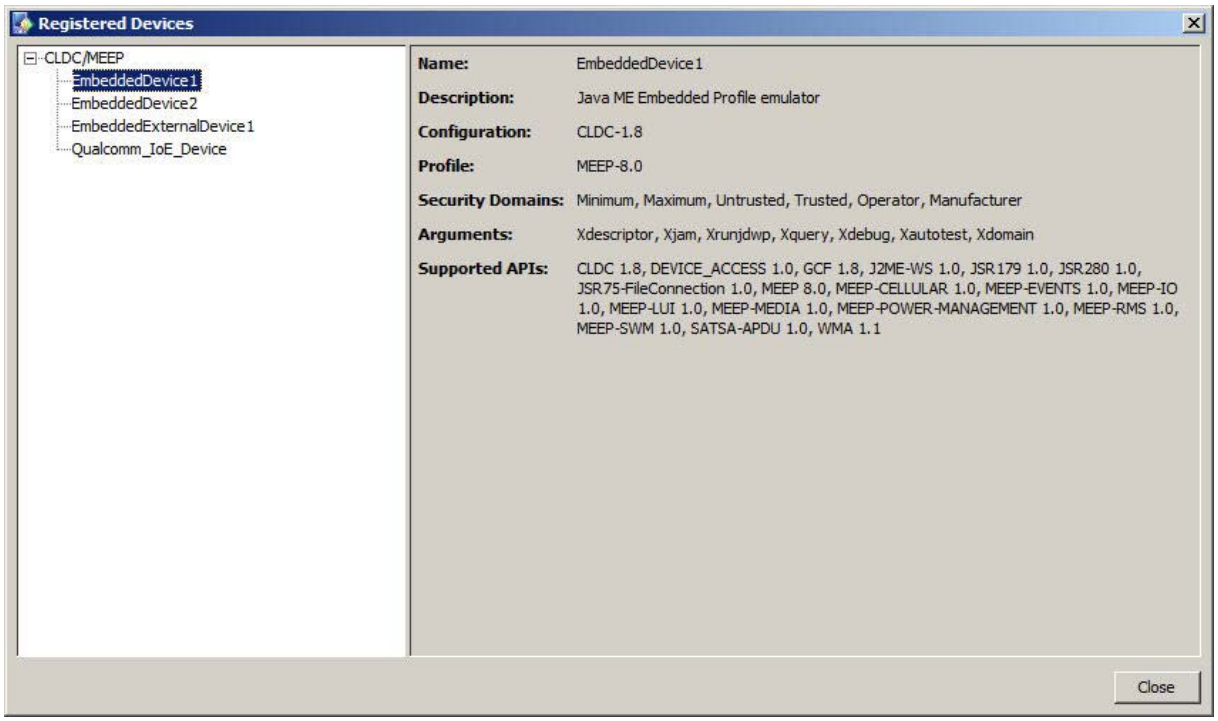
5. Enter an **IP Address** or **Host Name** (for example, localhost). Click **OK**.

   If an embedded hardware device is attached, you can select a COM port for the hardware connection in the Select COM Port field.

If you have an address you no longer want to detect, select the address and click **Remove**. The device is no longer displayed in the Device Connections Manager.

To see a list of registered devices and their configuration information, right-click the Device Manager icon in the system tray and select **Registered Devices**, as shown in Figure 5–5.

Figure 5–5 The Registered Devices Screen



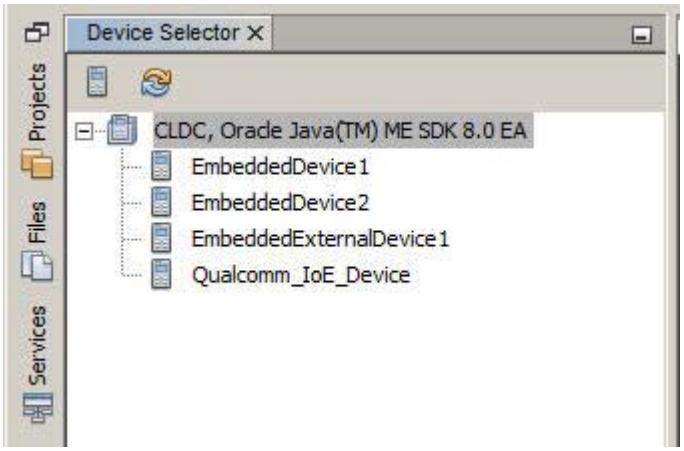
Using the Device Selector

The Device Selector lists the devices currently available in NetBeans 8.0 Beta, grouped by platform, as shown in Figure 5–6. If the Device Selector is not visible, click on the Device Selector tab, or select **Tools** > **Java ME** > **Device Selector**.

The list in the Device Selector matches the list displayed in the Registered Devices window, as shown in Figure 5–6.

> **Note:** The `ExternalEmbeddedDevice1` only appears as a configured device when an external hardware device is detected, such as a Raspberry Pi or Qualcomm Orion IoE (MB997B) embedded board.

Figure 5–6 The Device Selector

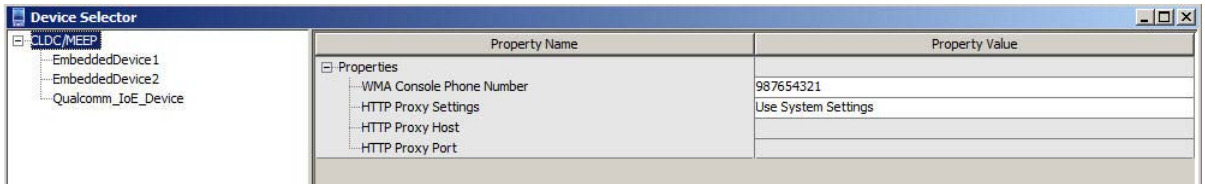


The Device Selector can also be launched from the Windows command line. Enter the following:

C:\>**Java_ME_platform_SDK_8.0_EA\bin\device-selector.exe**

This displays the standalone Device Selector, as shown in Figure 5–7.

Figure 5–7 The Standalone Device Selector



Viewing Properties

Using the Device Selector, you can view properties information for a platform or a device.

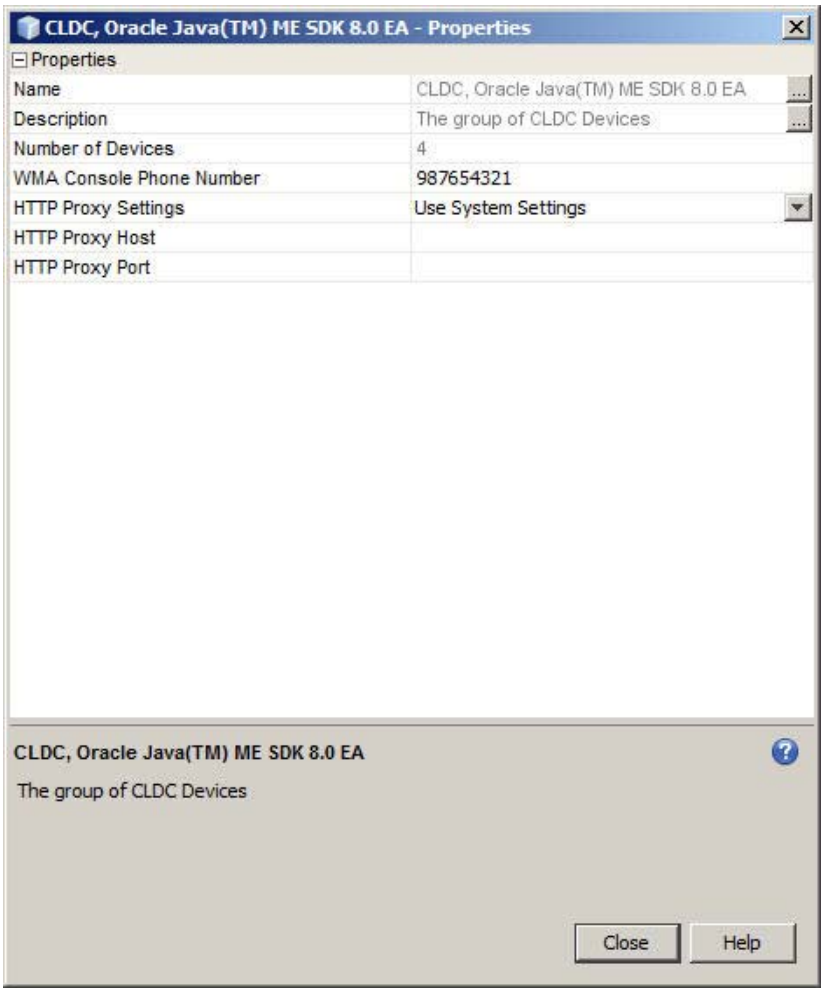
The Properties window is, by default, located on the right side of the NetBeans 8.0 Beta user interface. If the Properties window is not visible, select **Windows > IDE Tools >Properties**

Instances of similar devices (for example, EmbeddedDevice1 and EmbeddedDevice2) have the same properties and capabilities, but unique names and phone numbers. This makes it easy for you to test communication between devices of the same type.

Viewing Platform Properties

To view platform properties from the Device Selector, click on the platform name (for example, CLDC, Oracle Java ME SDK 8 EA 2). To view the platform properties in a separate window, right-click the platform node and select Properties. This displays the Platform Properties in a pop-up window, as shown in Figure 5–8.

Figure 5–8 Viewing Platform Properties

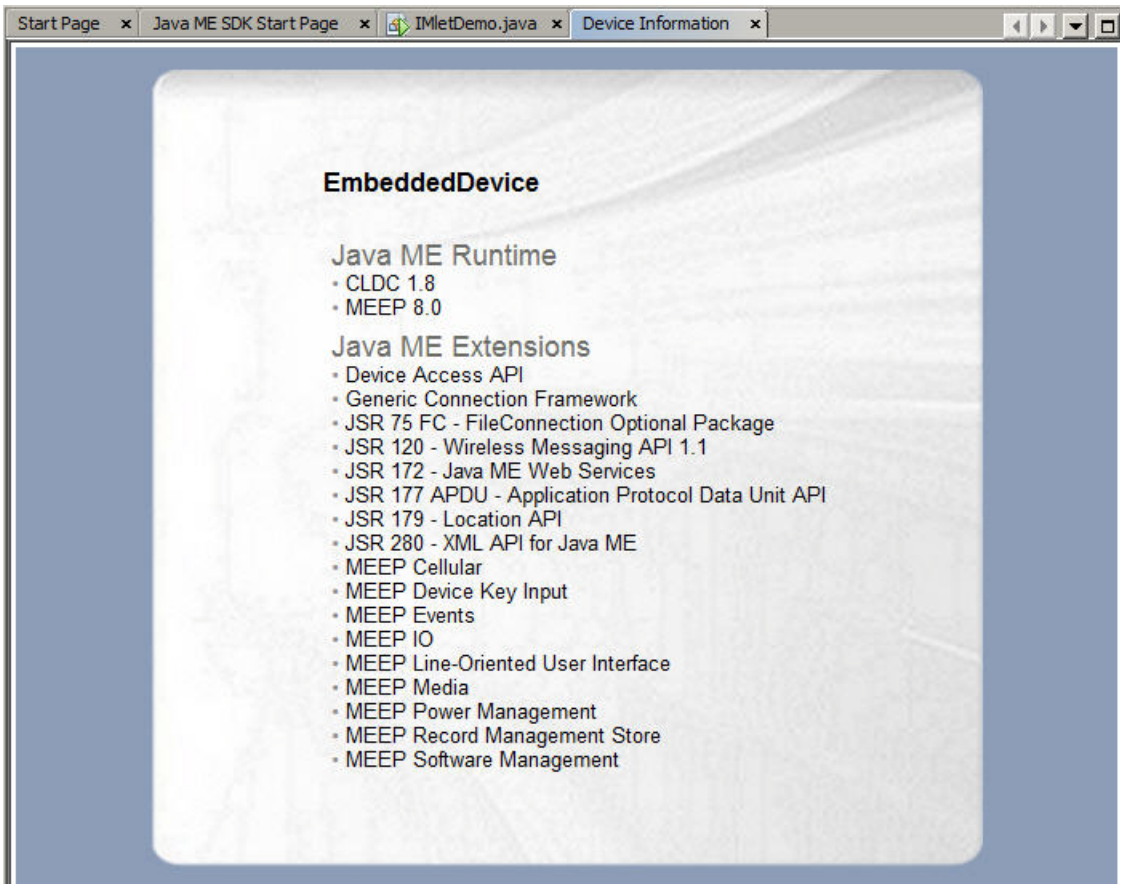


Viewing Device Information

In the Device Selector window, double-click on a device name (for example, EmbeddedDevice1). The Device Information tab opens in the Main window.

The Device Information window displays details about the device, such as the supported runtime, supported Java ME extensions, supported optional packages, and other capabilities, such as power management or cellular support, as shown in Figure 5–9.

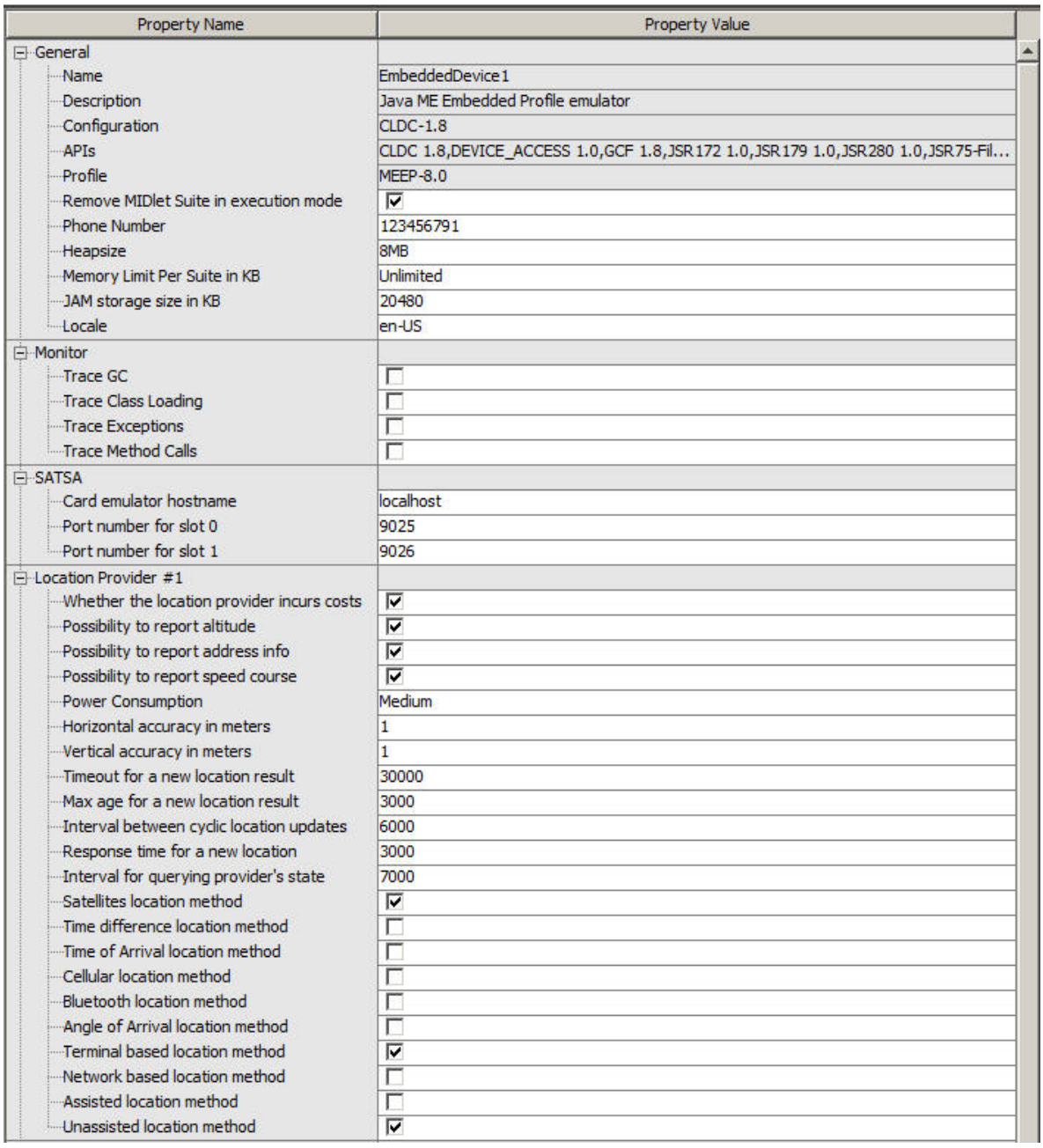
Figure 5–9 Viewing Device Information



Viewing Device Properties

In the Device Selector window, click a device name to display the properties for the device, as shown in Figure 5–10.

Figure 5–10 Viewing Device Properties



Property Name	Property Value
General	
Name	EmbeddedDevice1
Description	Java ME Embedded Profile emulator
Configuration	CLDC-1.8
APIs	CLDC 1.8,DEVICE_ACCESS 1.0,GCF 1.8,JSR172 1.0,JSR179 1.0,JSR280 1.0,JSR75-Fil...
Profile	MEEP-8.0
Remove MIDlet Suite in execution mode	☑
Phone Number	123456791
Heapsize	8MB
Memory Limit Per Suite in KB	Unlimited
JAM storage size in KB	20480
Locale	en-US
Monitor	
Trace GC	☐
Trace Class Loading	☐
Trace Exceptions	☐
Trace Method Calls	☐
SATSA	
Card emulator hostname	localhost
Port number for slot 0	9025
Port number for slot 1	9026
Location Provider #1	
Whether the location provider incurs costs	☑
Possibility to report altitude	☑
Possibility to report address info	☑
Possibility to report speed course	☑
Power Consumption	Medium
Horizontal accuracy in meters	1
Vertical accuracy in meters	1
Timeout for a new location result	30000
Max age for a new location result	3000
Interval between cyclic location updates	6000
Response time for a new location	3000
Interval for querying provider's state	7000
Satellites location method	☑
Time difference location method	☐
Time of Arrival location method	☐
Cellular location method	☐
Bluetooth location method	☐
Angle of Arrival location method	☐
Terminal based location method	☑
Network based location method	☐
Assisted location method	☐
Unassisted location method	☑

Setting Device Properties

In the Device Selector window, right-click a device and select **Properties**. Any properties shown in gray font cannot be changed. You can adjust the device properties shown in black. Only MEEP options can be adjusted.

Setting General Properties

This section lists general properties that can be changed.

Remove IMlet Suite in execution mode. If this option is enabled, record stores and other resources created by the IMlet are removed when you exit the IMlet (assuming the IMlet was started in execution mode).

Phone Number. You can set the phone number to any appropriate sequence, considering country codes, area codes, and so forth. If you reset this value, the setting applies to future instances. The number is a base value for the selected device.

Heapsize. The heap is the memory allocated on a device to store your application objects. The Heapsize property is the maximum heap size for the emulator. You can select a new maximum size from the drop down menu.

Memory Limit Per Suite. This property allows you to define how much memory an IMlet suite is allocated upon launch. Unlimited means the IMlet can use as much memory as required to run.

JAM storage size in KB. The amount of space available for applications installed over the air.

Locale. Type in the locale as defined in the ME Embedded Profile specification at:

`https://jcp.org/aboutJava/communityprocess/edr/jsr361/index.html`

Setting Monitor Properties

If enabled (checked), the Check boxes for Trace GC (garbage collection), Trace Class Loading, Trace Exceptions, and Trace Method Calls activate tracing for the selected device the next time the emulator is launched. The trace output is displayed at runtime in the user interface Output window.

> **Note:** Trace Method Calls returns many messages, and emulator performance can be affected.

Setting SATSA Properties

Security and Trust Services (SATSA) provide the ability to define security settings. You can define the hostname of a Java Card emulator, and port numbers for slots `0` and `1`.

Setting Properties for Location Provider #1 and #2

These properties determine the selection of a location provider. Two providers are offered so that your application can be tested matching the location provider criteria.

If you select a property a short explanation is shown in the description area just below the Properties table. For more information on these values, see the Location API at:

`http://jcp.org/en/jsr/detail?id=179`.

Setting Landmark Editor Properties

The Landmark Editor has only one property, maximum length of input.

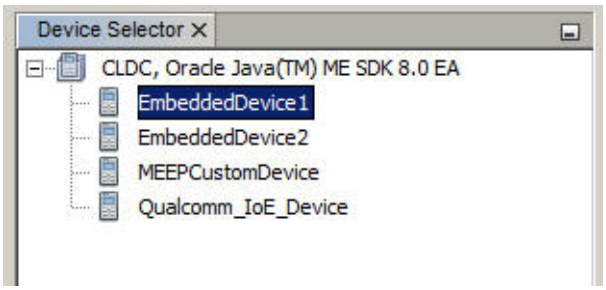
Setting Security Properties

Security properties for a device are set at the time the device is created, using the Client Security Model.

To edit security settings for a specific device in the Device Selector:

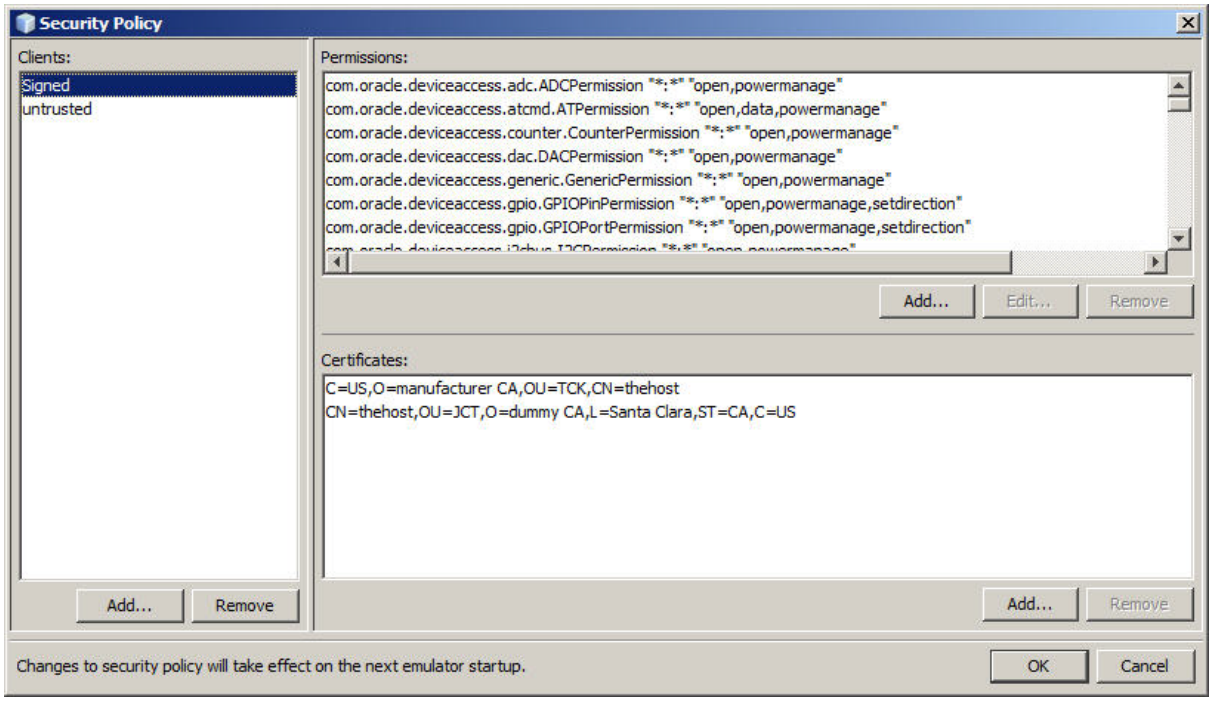
1. Select a device from the list of available devices, for example, **EmbeddedDevice1**, as shown in Figure 5–11.

Figure 5–11 The Device Selector Screen with a Selected Device



2. Right click on the selected device to display the device menu. Scroll down and select **Edit Security Policy**.
3. This displays the Security Policy screen, as shown in

Figure 5–12 The Security Policy Screen

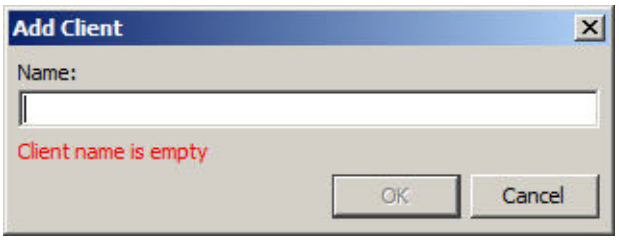


4. `EmbeddedDevice1` is a Signed device. The specific permissions that are granted to it are defined in the `Permissions` category. The certificates that verify its signing are listed in the `Certificates` category.

   To edit the `EmbeddedDevice1` security settings, do the following:

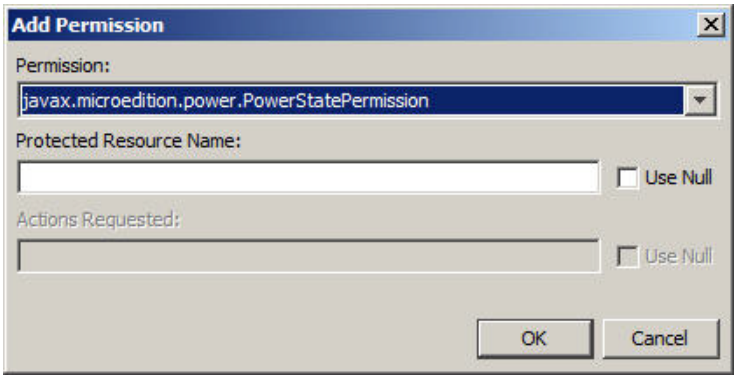
   a. To add a Client, click **Add** in the Clients category. This displays the `Add Client` form, as shown in Figure 5–13. Enter the new Client name and click **OK**.

Figure 5–13 The Add Client Form



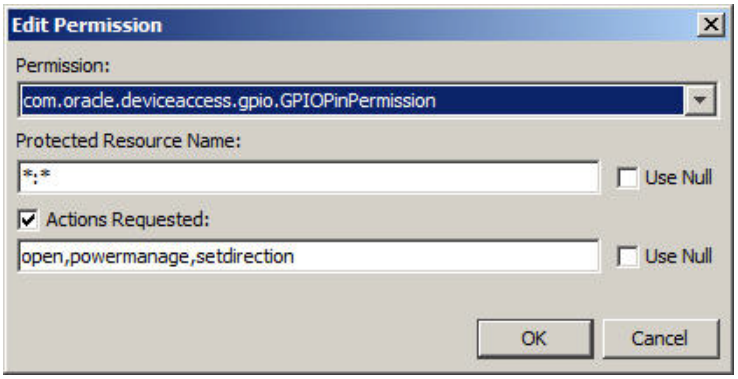
b. To add a Permission, click **Add** in the Permissions category. This displays the `Add Permissions` form, as shown in Figure 5–14. Select a permission from the drop down list in the Permission field, enter a Protected Resource Name, and click **OK**.

Figure 5–14 The Add Permission Form



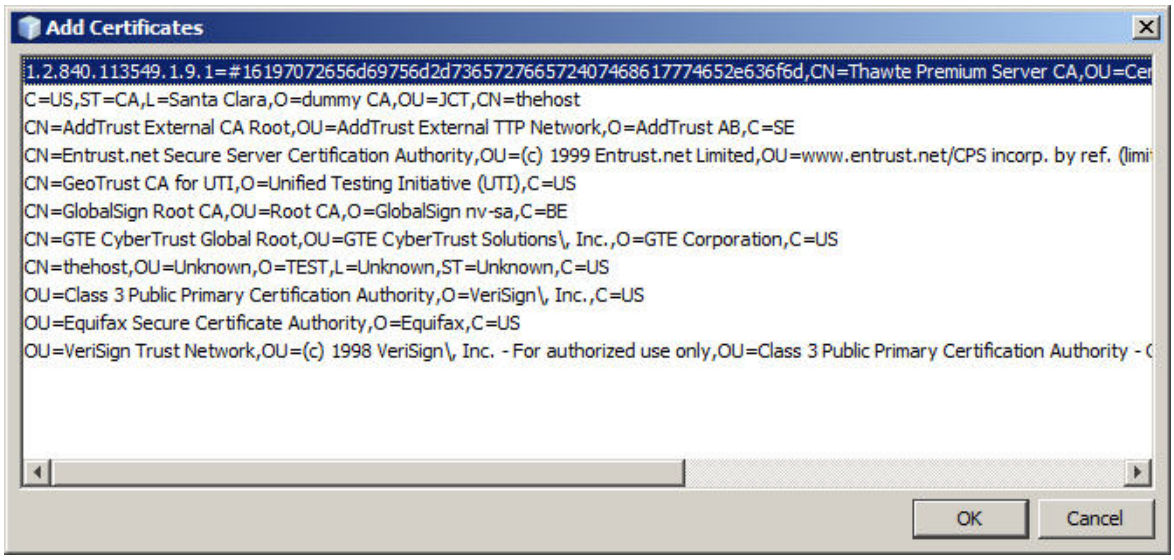
c. To edit a Permission, click **Edit** in the Permissions category. This displays the `Edit Permissions` form, as shown in Figure 5–15. Select a permission from the drop down list in the Permission field, and make other edits, for example, to the Actions Requested list, and click **OK**.

Figure 5–15 The Edit Permission Form



d. To add a Certificate, click **Add** in the Certificates category. This displays the `Add Certificates` form, as shown in Figure 5–16. Enter the certificate information and click **OK**.

Figure 5–16 The Add Certificates Form



e. Once you have finished making Security Policy changes, click **OK**. The next time you start the configured device, your changes will take effect.

Click **Remove** in any category to remove a Client, Permission, or Certificate, and click **OK** for it to take effect.

For more detailed information on the Security Client Model, see the Java ME Embedded Profile (MEEP) Specification, at:

`https://jcp.org/aboutJava/communityprocess/edr/jsr361/index.html`

Using the Custom Device Editor

With the Custom Device Editor you can create your own devices. The appearance of a custom device is generic, but the functionality can be tailored to your own specifications.

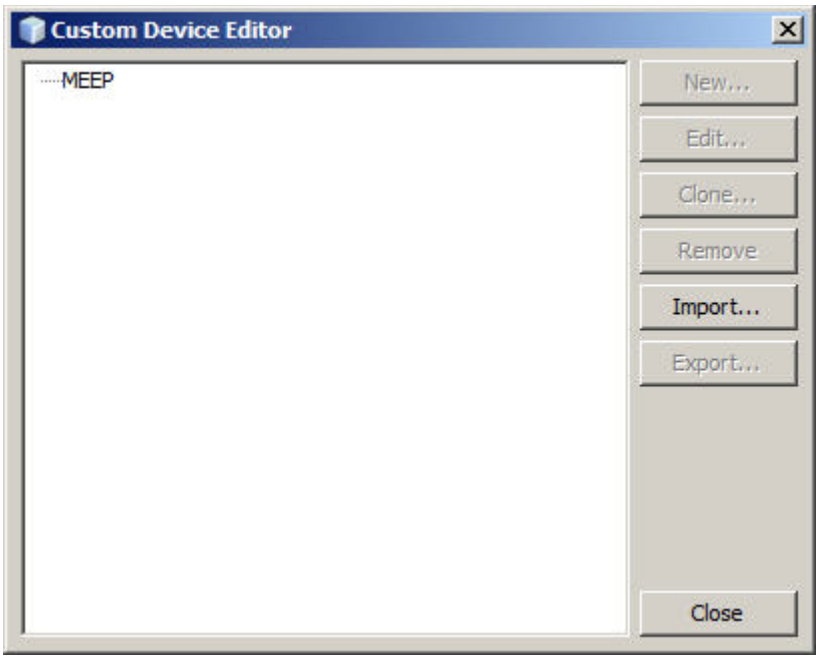
Creating a Custom Device

Follow these steps to create a custom device in NetBeans 8.0 Beta:

1. Select **Tools** > **Java ME** > **Custom Device Editor**

   The Custom Device Editor is shown in Figure 5–17. The custom device tree displays platforms and custom devices, if any. For example, the ME Embedded Platform (MEEP).

Figure 5–17 The Custom Device Editor

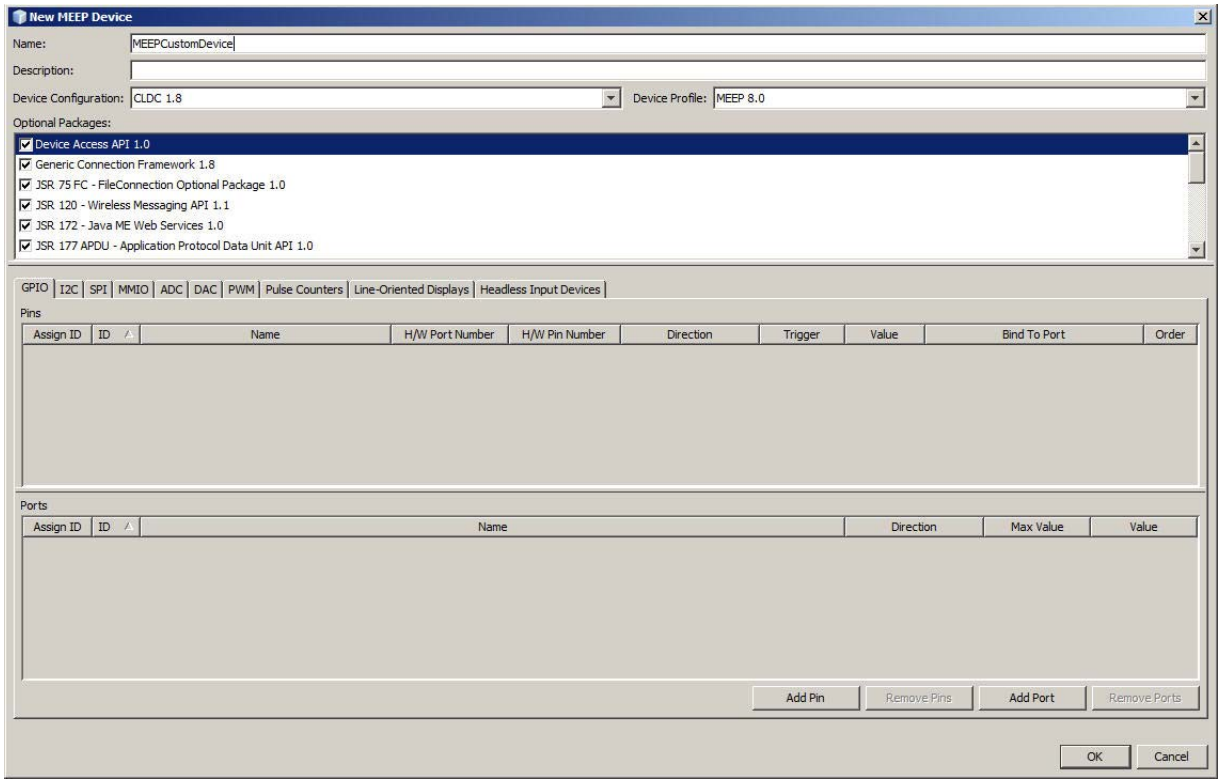


To run a standalone Custom Device Editor, use the `device-editor.exe` command from the Windows command line. For example:

C:>**Java_ME_platform_SDK_8.0_EA\bin\device-editor.exe**

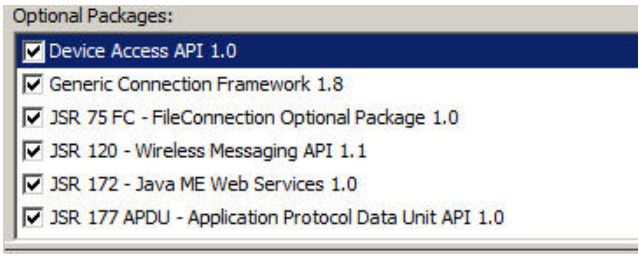
2. Select a platform and click the **New...** button. This displays the New Device dialog box, as shown in Figure 5–18.

Figure 5–18 The New MEEP Device Box



3. Change the default **Device Configuration** or **Device Profile** to match the specifications for your new device.
4. Select **Optional Packages**. Optional packages provide additional functionality to your device and can be individually selected to correspond to the configuration of your custom device, as shown in Figure 5–19.

Figure 5–19 New Custom Device Optional Packages



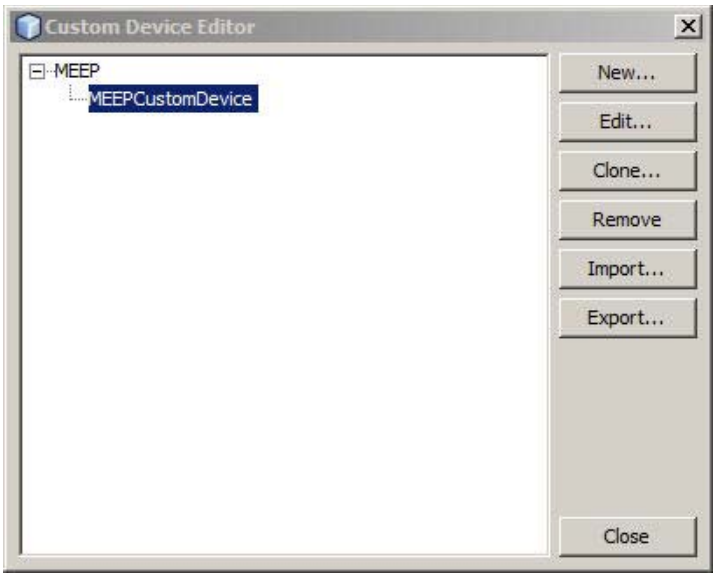
5. Define properties for the interfaces and protocols supported by your custom device, as shown in Figure 5–20. For more information on custom device protocols and property settings, see "Setting Custom Device Properties."

Figure 5–20 New Custom Device Protocols

6. Click **OK**.

   Your device is added to the custom device tree, as shown in Figure 5–21.

Figure 5–21 A Newly-Created Custom Device



Eventually, the newly-added device appears in the Device Selector. You can run projects from the IDE or from the command line for the custom device.

If you do not want a custom device to appear in the Device Selector, you must remove it from the custom device tree.

Once a newly created custom device is added to the Device Selector, the device definition is saved in:

<*install_dir*>`\toolkit-lib\devices`

Setting Custom Device Properties

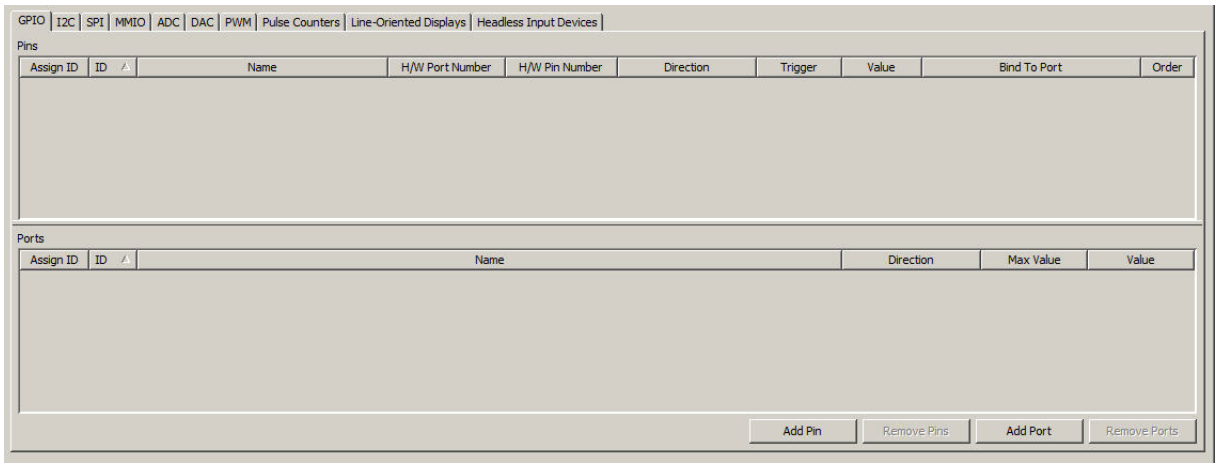
When you create a new embedded device using the Custom Device Editor (**Tools** > **Java ME** > **Custom Device Editor**) you can use the default implementation or create your own custom implementation for the interfaces discussed in this section.

You can set device properties when you first create your custom device. Or, you can define device properties later, by selecting your custom device in the Custom Device Editor, as shown in Figure 5–21, and clicking **Edit**.

General Purpose Input Output (GPIO)

A GPIO port is a platform-defined grouping of GPIO pins that may be configured for output, input, or bidirectional. The custom device GPIO screen is shown in Figure 5–22.

Figure 5–22 The GPIO Custom Device Screen



To add a custom Pin:

1. Click **Add Pin** to add a row to the Pins table.
2. A new row is highlighted in the Pins table. To customize a field in the pin row, double-click in a cell and define the field for your custom device.
3. For additional pins, click **Add Pin** again. A second row is added, allowing you to define a second pin for your custom device. Add as many rows as required for the number of pins defined for your custom device.

To add a custom Port:

1. Click **Add Port** to add a row to the Ports table.
2. A new row is highlighted in the Ports table. To customize a field in the port row, double-click in a cell and define the field for your custom device.
3. For additional ports, click **Add Port** again. A second row is added, allowing you to define a second port for your custom device. Add as many rows as required for the number of ports defined for your custom device.

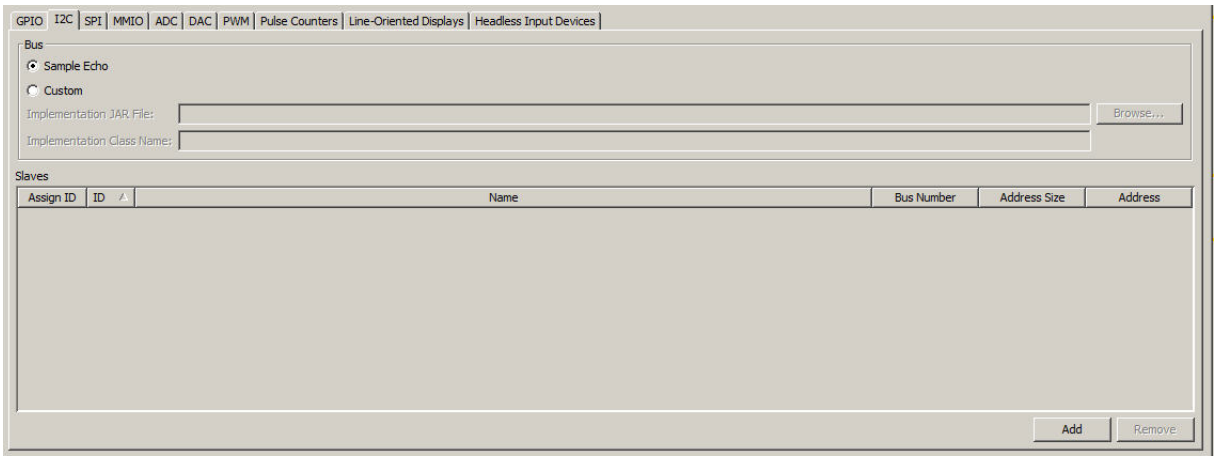
To remove a pin, highlight the row for the pin to be removed, and click **Remove Pins**. To remove a port, highlight the row for the port to be removed, and click **Remove Ports**.

When your custom device is fully configured, click **OK** to add it to the Custom Device Editor, as shown in Figure 5–21.

Inter-Integrated Circuit (I2C) and Serial Peripheral Interface (SPI)

The Inter-Integrated Circuit (I2C) and Serial Peripheral Interface (SPI) are very similar. Each simulates a simple peripheral slave device that echoes back data sent to it. The I2C and SPI custom screens are also very similar, as shown in Figure 5–23.

Figure 5–23 The I2C or SPI Custom Device Screen



To create a custom I2C or SPI interface for your custom device:

1. Select **Sample Echo** to select the default bus. Or, select **Custom** to define your own bus implementation.

   If you select **Custom**:

   - Supply your bus implementation JAR file and the name of the Java class that implements the bus.

     For I2C, the bus is:

     ```
     com.oracle.jme.toolkit.deviceaccess.i2c.I2CSlaveBus
     ```

     For SPI, the bus is:

     ```
     com.oracle.jme.toolkit.deviceaccess.spi.SPISlaveBus
     ```

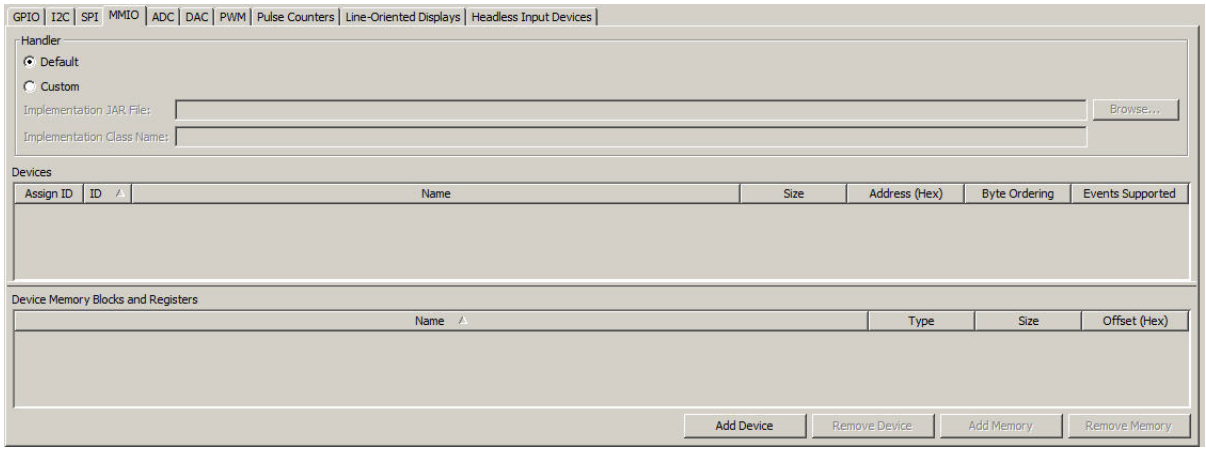
2. To add Slaves, click **Add** and specify an ID and Name.
3. When your I2C or SPI device is defined, click **OK**. This adds the I2C or SPI implementation to your custom device.

To remove a row, select the row and click **Remove**.

Memory-Mapped I/O (MMIO)

The MMIO tab emulates the Memory-Mapped I/O (MMIO) interface bus. It facilitates input/output between CPU memory and a peripheral device. To define a custom MMIO implementation use the MMIO custom device screen, as shown in Figure 5–24.

Figure 5–24 The MMIO Custom Device Screen



If you want to provide your own MMIO emulation, you must specify a custom handler.

To create a custom MMIO implementation:

1. Click **Default** to use the default handler JAR file. If you click **Custom**, you must specify a custom handler using the following fields:
   - **Implementation JAR File**. Supply a JAR file for your implementation.
   - **Implementation Class Name**. Supply the name of the Java class that implements `com.oracle.jme.toolkit.deviceaccess.mmio.MMIOHandler`.

   The default handler JAR file is:

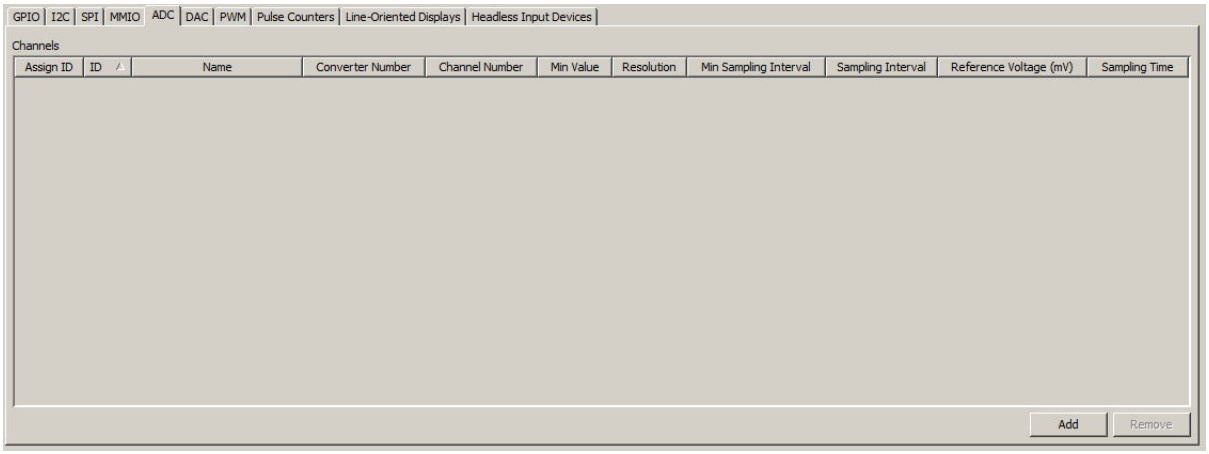
   installdir`\toolkit-lib\devices\Embeddedevice1\code\emulator_deviceaccess_mmio-sample-handler.jar`
2. Click **Add Device**. This adds a new row to the Devices table.
3. To define individual settings in a row for your custom device, double-click in a cell to enter a value for that field.
4. To add memory, click **Add Memory**. This adds a row to the Device Memory Blocks and Registers table.
5. When you are satisfied with your custom entries, click **OK**

To remove a device, select a row and click **Remove Device**. To remove memory, select a row and click **Remove Memory**.

Analog to Digital Conversion (ADC)

Analog to digital conversion takes an analog stream as input and converts it to a sequence of digital numbers. To specify a custom analog-to-digital conversion channel, you must add a channel to the Channels window, as shown in Figure 5–25.

Figure 5–25 The ADC Custom Device Screen



To add a custom ADC channel:

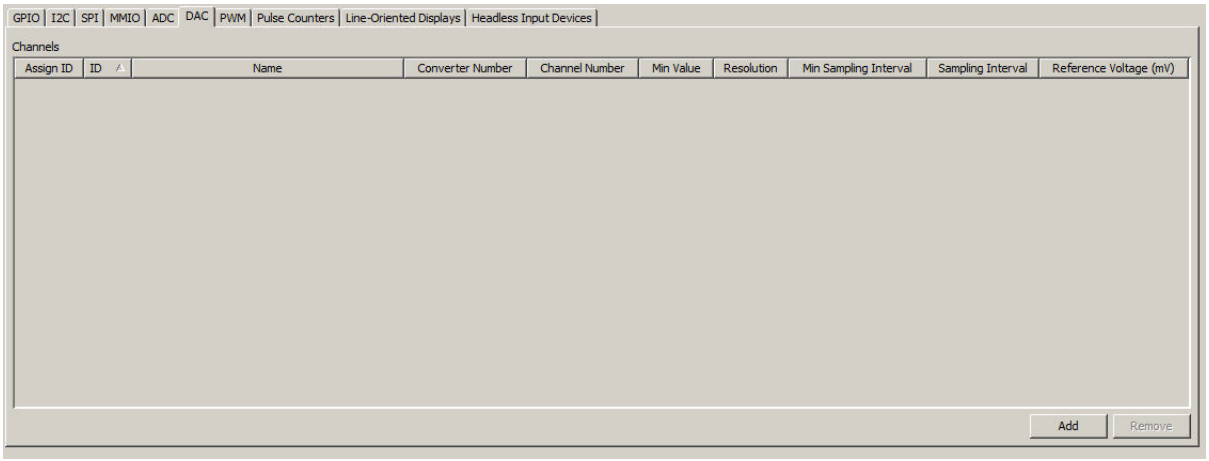
1. Click **Add**. This adds a new row to the Channels table.
2. To define individual settings in a row for your custom device, double-click in a cell to enter a value for that field.
3. When you are satisfied with your custom entries, click **OK**

To remove a channel, select a row and click **Remove**.

Digital to Analog Conversion (DAC)

Digital conversion takes a sequence of numbers as an input and converts it to an analog stream. To specify a custom digital-to-analog conversion channel, you must add a channel to the Channels window, as shown in Figure 5–25.

Figure 5–26 The DAC Custom Device Screen



To add a custom DAC channel:

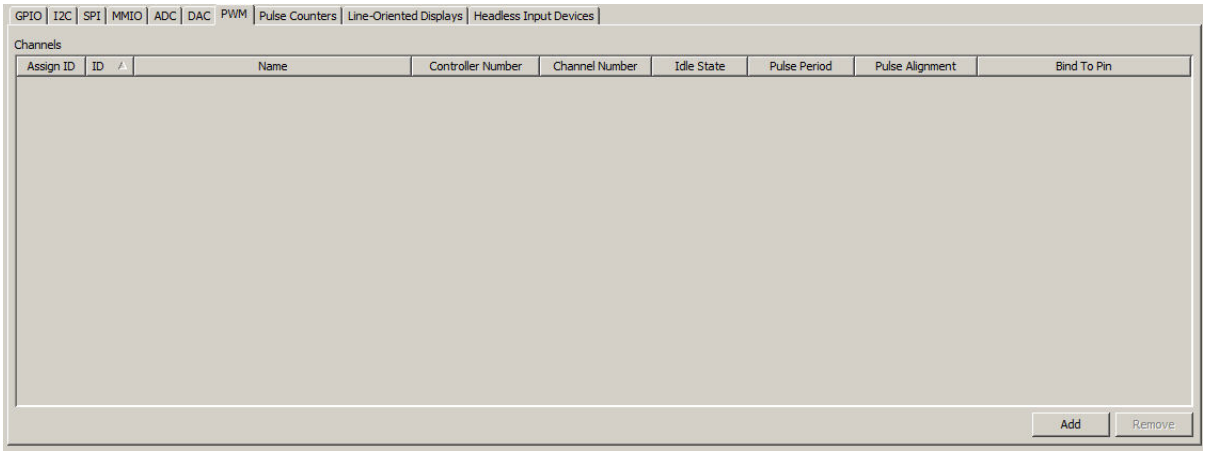
1. Click **Add**. This adds a new row to the Channels table.
2. To define individual settings in a row for your custom device, double-click in a cell to enter a value for that field.
3. When you are satisfied with your custom entries, click **OK**

To remove a channel, select a row and click **Remove**.

Pulse Width Modulation (PWM)

Pulse width modulation conforms the width of a digital signal based on its duration. To specify a custom pulse width modulation channel, you must add a channel to the Channels window, as shown in Figure 5–27.

Figure 5–27 The PWM Custom Device Screen



To add a custom PWM channel:

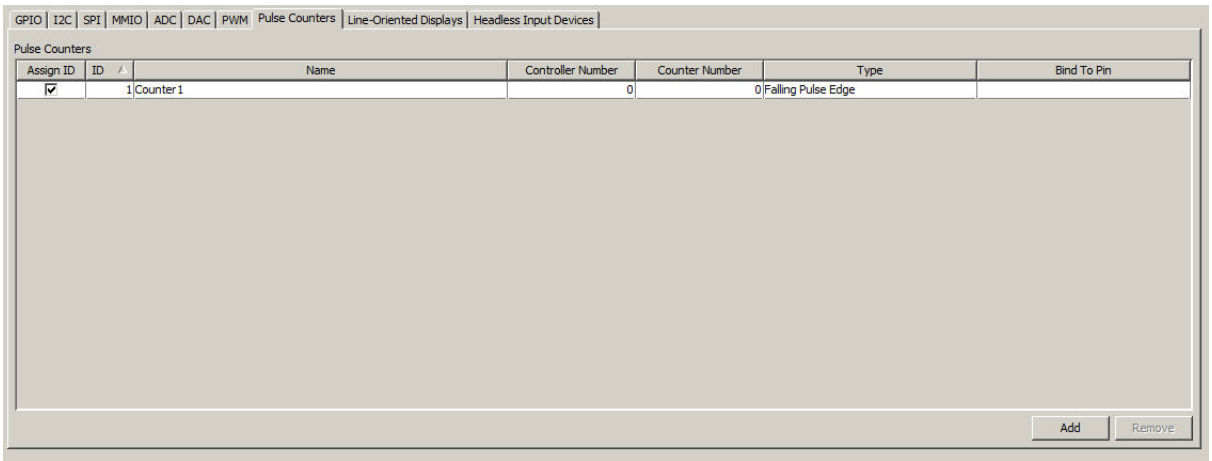
1. Click **Add**. This adds a new row to the Channels table.
2. To define individual settings in a row for your custom device, double-click in a cell to enter a value for that field.
3. Once you are satisfied with your custom entries, click **OK**

To remove a channel, select a row and click **Remove**.

Pulse Counters

A pulse counter tracks the number of pulses sent to a device. To specify a custom pulse counter, you must add a channel to the Pulse Counters window, as shown in Figure 5–28.

Figure 5–28 The Pulse Counters Custom Device Screen



To add custom Pulse Counter:

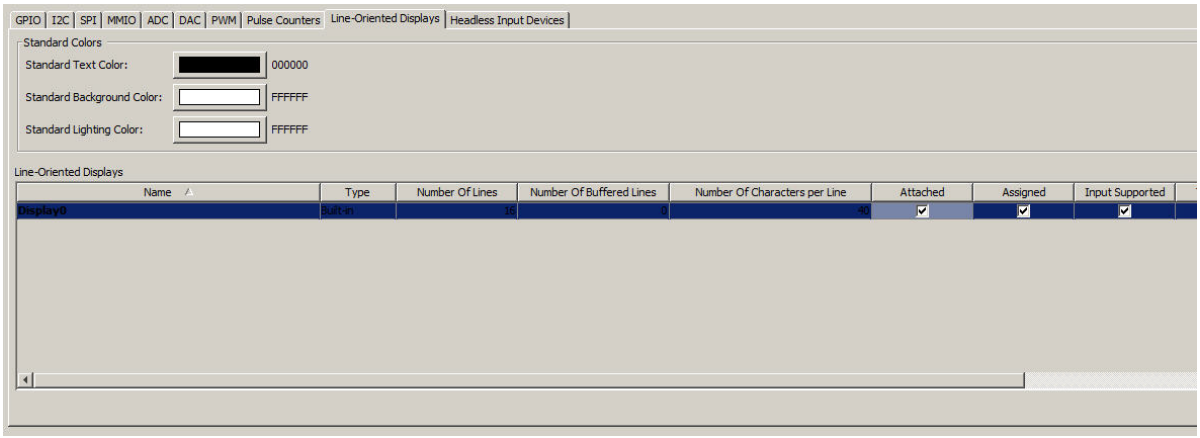
1. Click **Add.** This adds a new row to the Pulse Counters table.
2. To define individual settings in a row for your custom device, double-click in a cell to enter a value for that field.
3. Once you are satisfied with your custom entries, click **OK**

To remove a channel, select a row and click **Remove**.

Line-Oriented Displays

Configuring a line-oriented display allows you to customize a simple user interface in emulation. To customize a line-oriented display, use the Line-Oriented Displays tab in the Custom Device Editor, as show in Figure 5–29.

Figure 5–29 The Line-Oriented Display Custom Device Screen



To configure a new Line-Oriented Display custom device:

1. Set the following **Standard Colors**:
   - **Standard Text Color**. The default color for text is black (`000000` in Hexadecimal). To change the text color, click on the rectangle. This displays the Standard Text Color form, as shown in Figure 5–30.

     To set text to a specific color, select the color square from the color map and click **OK**. Or, click on the following tabs to customize text color based on a variety of sliding-scale color values:

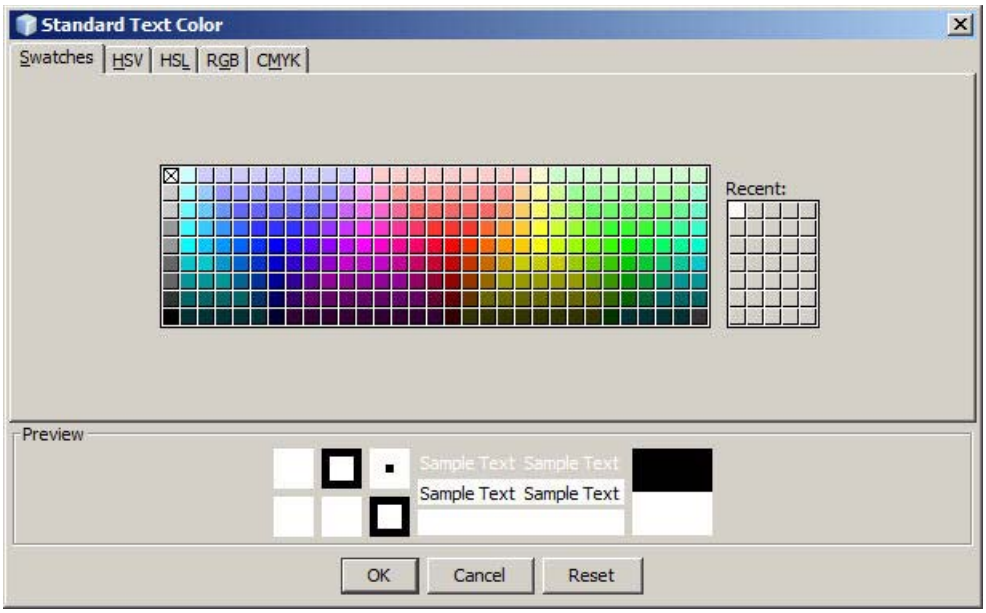
     – **HSV**. Allows you to customize Hue, Saturation, and Value fields.

     – **HSL**. Allows you to customize Hue, Saturation, and Lightness fields.

     – **RGB**. Allows you to customize Red, Green, and Blue fields.

     – **CMYK**. Allows you to customize Cyan, Magenta, and Yellow fields.

Figure 5–30 The Standard Text Color Form



- **Standard Background Color**. The default background color for is white (FFFFFF in Hexadecimal). To change the background color, click on the rectangle. This displays the Standard Background Color form (same as the Standard Text Color form).

  Select a background color and click **OK**. When the color is changed, the new color value is displayed in Hex.

- **Standard Lighting Color**. The default lighting (highlighting) color is white (FFFFFF in Hexadecimal). To change the lighting color, click on the rectangle. This displays the Standard Lighting Color form (same as the Standard Text Color form).

  Select a lighting color and click **OK**. When the color is changed, the new color value is displayed in Hex.

2. Click **Add.** This adds a new row to the Line-Oriented Displays table.
3. To define individual settings in a row for your custom device, double-click in a cell to enter a value for that field.

> **Note:** The Line-Oriented Display Custom Device Screen has many settings that scroll to the right. To fully configure your custom implementation, ensure that all the cells in the new row have been defined.

4. Once you are satisfied with your custom entries, click **OK**

To remove a Line-Oriented Displays, select a row and click **Remove**.

Headless Input Devices

A headless input device could be buttons, for example, on an embedded hardware device, or other entry devices not connected to a monitor or screen. To add a custom Headless Input Device, do the following:

1. Click **Add.** This adds a new row to the Headless Input Devices table.
2. To define individual settings in a row for your custom device, double-click in a cell to enter a value for that field.
3. Once you are satisfied with your custom entries, click **OK**

To remove a channel, select a row and click **Remove**.

Managing Custom Devices

Custom devices should always be managed using the Custom Device Editor. Using the tool ensures that your device can be detected and integrated with the Oracle Java ME SDK 8 EA 2.

- **New**. Select a platform and click **New** to add a new device.
- **Edit**. Select a device to change, and click **Edit**.
- **Clone**. Select a device to copy, and click **Clone**. To prevent confusion, be sure to provide a unique name.
- **Remove**. Select a device to delete and click **Remove**. This action completely deletes the device.
- **Import**. Select a `.zip` file created with the Export command and click Import.
- **Export**. Select one or more devices to save, and click **Export**.

  An exported device is stored in a `.zip` file and saved to any location the user chooses.

Making Device Connections

The configuration of all peripherals, except UART, can be inspected in the emulator main window. The configuration of UART is defined by the hardware configuration of the COM ports on your Windows 7 PC.

For more information on connecting to a UART Device, see "Connecting to a UART Device."

To open a serial port, such as COM1 or COM2, in Windows you can use the Device Connections Manager. In application code, you can use `Connector.open("comm:COM1")`

Connecting to a UART Device

To utilize the Universal Asynchronous Receiver/Transmitter (UART) functionality, you need to create a configuration file as shown here:

```
javacall\configuration\ojwc\win32\properties_deviceaccess_at.xml
```

The configuration of the hardware device name for the UART ports on your Windows 7 PC is made up of two settings:

- The `deviceNumber` property of `UARTDeviceConfiguration`
- The system property of `device.uart.prefix`

For example, if `UARTDeviceConfiguration.deviceNumber = 1` and `device.uart.prefix = COM`, then the hardware device name is `COM1`.

For an already opened UART, you can add it to the peripheral manager by calling `PeripheralManager.open` with the configuration object.

Additional Peripherals

The Oracle Java ME SDK 8 EA 2 `EmbeddedDevice1` emulator provides support for additional peripherals, ATDevice and Watchdog timers. Use the following configuration settings to open them using the PeripheralManager:

- **Watchdog Timers**. There are two Watchdog timers on the `EmbeddedDevice1` emulator with the following configurations:
  - **Device Name**: WDG, ID: 30, Hardware Timer's Number: 1 (This is a regular watchdog timer.)
  - **Device Name**: WWDG, ID: 31, Hardware Timer's Number: 2 (This is a windowed watchdog timer).

  Watchdog timers provide the services to track how your application functions. Because no hardware is available on win32, all running applications are killed when the Watchdog event is fired.
- **ATDevice.** ATDevice is a simple AT commands-based device on the `EmbeddedDevice1` emulator that responds "Ok" to any command. It has the following configuration:
  - **Device Name**. EMUL, ID: 13, Device Number: 1, Hardware Channel's Number: 1

6

Using Sample Applications

The Oracle Java ME SDK 8 EA 2 sample applications introduce you to the emulator's API features and the Oracle Java ME SDK 8 EA 2 features, tools, and utilities that support the various APIs.

> **Note:** Before using the Oracle Java ME SDK 8 EA 2 sample applications, see "Installation and Runtime Security Guidelines in Appendix B, "Installation and Runtime Security Guidelines." Some demos use network access and open ports, and do not include protections against malicious intrusion. If you choose to run the sample projects, you should ensure your environment is secure.

Installing Sample Applications

Sample applications are installed using an Oracle Java ME SDK 8 EA 2 zip file, `jmesdk-8_0-ea2-samples-<build_number>-<date>.zip`. The default location for installing the sample applications is in the location where you have installed Oracle Java ME SDK 8 EA 2. Do the following:

1. Download the sample applications file from the Oracle Technology Network (OTN) to your machine.
2. Move the sample applications file into the location where you have installed Oracle Java ME SDK 8 EA 2. For example:

   ```
   C:\Java_ME_platform_SDK_8.0_EA
   ```

3. Unzip the sample applications file. This displays a second zip file, `com.oracle.javame.sdk.sample.applications.zip`.
4. Unzip the `com.oracle.javame.sdk.sample.applications.zip` file. This creates the /apps directory, as shown here:

   ```
   C:\Java_ME_platform_SDK_8.0_EA\apps
   ```

5. Change to the /apps directory. You see the following directories listed, one for each sample application:
   - DataCollectionDemo
   - GPIODemo
   - I2CDemo
   - LightTrackDemo
   - NetworkDemo

- PDAPDemo
- SystemControllerDemo

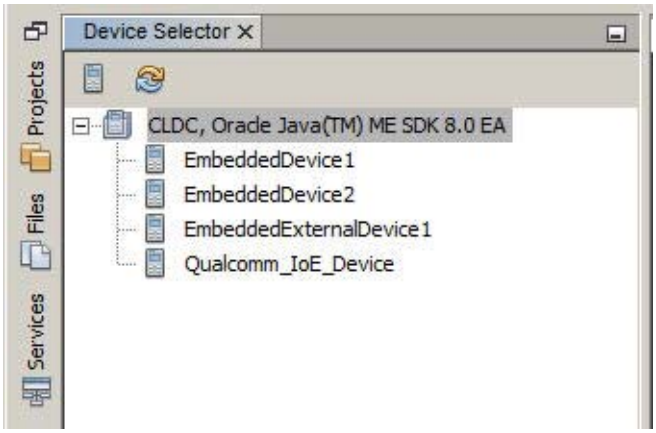
Configuring the Web Browser and Proxy Settings

If you are behind a firewall, you can configure the sample applications to use proxy server settings that you define.

The sample application proxy server settings typically match the proxy server settings used in your web browser. To manually set the proxy server settings for your sample applications, do the following:

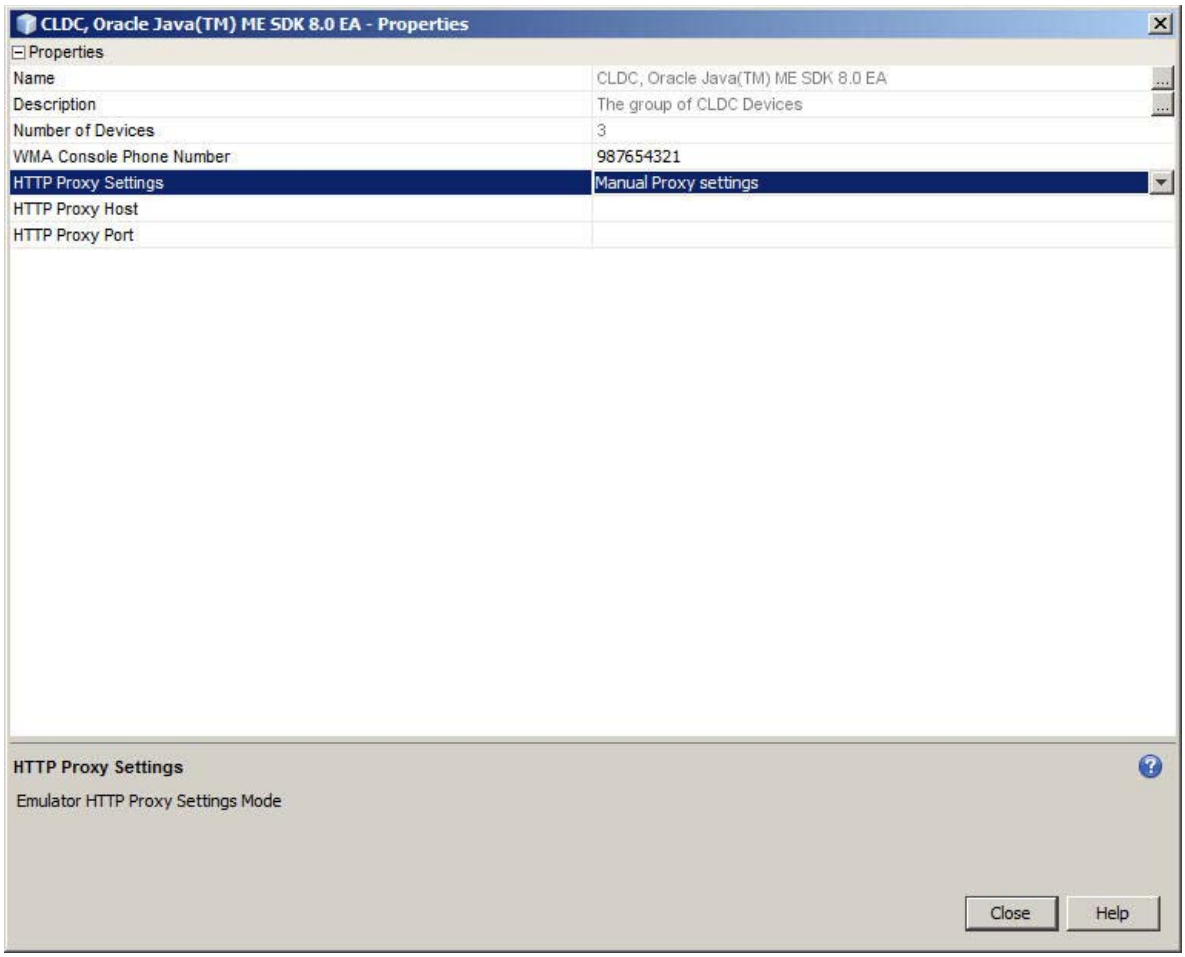
1. Select **Tools** > **JavaME** > **Device Selector** to display the Device Selector in NetBeans 8.0 Beta, as shown in Figure 6–1.

Figure 6–1 The Device Selector



2. Right click on **CLDC, Oracle Java(TM) ME SDK 8.0 EA** and select **Properties** from the pop-up menu, to display the Device Selector properties screen, as shown in Figure 6–2.

Figure 6–2 The Device Selector Properties Screen



3. To configure the Device Selector Properties proxy settings, fill in the **HTTP Proxy Settings**, **HTTP Proxy Host**, and **HTTP Proxy Port** fields to match your network and browser settings.

Running Sample Applications

This section describes how to use sample applications created specifically for Oracle Java ME SDK 8 EA 2. Because these sample applications are headless, the only user interface is to observe application status in the emulator's External Events Generator, in the Output window, or in the console window, if you execute the demo from the command line.

Running the Data Collection Demo

The Data Collection demo demonstrates the following functionality:

- Multiple virtual machines (MVM)
- Inter-IMlet communication using local datagrams
- Device Access API pulse counter
- Device Access API serial peripheral interface
- Logging API

In this demo, several data collector IMlets read data from peripheral devices using the Device Access API and send the data to a data processor.

For more information on the setup and behavior of the Data Collection demo, see the DataCollectionDemo `readme.txt` file, in the location where you have installed the Oracle Java ME SDK 8 EA 2 sample applications. For example:

```
C:\Java_ME_platform_SDK_8.0_EA\apps\DataCollectionDemo
```

For more information on the Qualcomm Orion IoE platform, see the *Oracle Java ME Embedded Getting Started Guide for the Reference Platform (Qualcomm Orion IoE)*.

Running the GPIO Demo

This demo can be run on an emulator. The implementations are different, as the emulator uses the external events generator, and the external device supports direct interaction.

Running the GPIODemo on the Emulator

1. Run GPIO demo on the **EmbeddedDevice1** emulator.
2. Click the **GPIO Pins** tab. This view approximates the device actions.
3. Click the **Tools** menu and select **External Events Generator** to open the External Events Generator, and click the **GPIO** tab.
4. Pressing **BUTTON 1** in the External Events Generator changes the state of the pin named BUTTON 1 in the EmbeddedDevice1 emulator. If he button value in the External Events Generator is changed to **High**, the button value in the EmbeddedDevice1 emulator is also changed to High.

Changing the state of a GPIO button is a toggle, which allows you to switch the pin value from Low to High, and back again.

Running the I2C Demo

This demo is designed to work with the Oracle Java ME SDK 8 EA 2. It has no user interaction.

Running the I2C Demo on the Emulator

1. Run the I2C demo on the **EmbeddedDevice1** emulator.
2. Click the **I2C** tab.
3. Using the Windows 7 user interface, navigate to the location where you have installed the I2C demo application. For example:

   C:`\Java_ME_platform_SDK_8.0_EA\apps\I2CDemo`
4. Double-click on the **I2CDemo.jad**.

   The demo acquires a slave named **I2C_Joystick**, writes data to the slave, and retrieves it. The demo is successful if the Sent Data and Received Data match.

Running the Network Demo

You can configure this demo as a server or as a client by editing the application descriptor. You launch two instances of this demo, the first one acts as a server and the second one acts as a client. The client instance attempts to connect to the server instance and if the connection is successful they exchange a message.

Running NetworkDemo on the Emulator

1. Create two instance projects of the **NetworkDemo** sample project.
2. Right click the first project and select **Properties**.
3. In the Platform category select the device **EmbeddedDevice1**. In the Application Description category set the value of the following property:

   ```
   Oracle-Demo-Network-Mode:Server
   ```

4. Click **OK**.
5. Launch the first project. It opens on the emulator `EmbeddedDevice1` and waits for a connection.
6. Right click the second project and select **Properties**.
7. In the Platform category select the device **EmbeddedDevice2**. In the Application Description category set the value of the following property:

   ```
   Oracle-Demo-Network-Mode:Client
   ```

8. Click **OK**.
9. Launch the second project. It opens on the emulator `EmbeddedDevice2`.
10. The client attempts to connect to the server. If successful, you see the following in the output tab of the first project (the server):

    ```
    Waiting for connection on port 5000
    Connection accepted
    Message received - Client messages
    ```

    The output of the second project (the client) shows the following:

    ```
    Connected to server localhost on port 5000
    Message received - Server string
    ```

Running NetworkDemo on the Reference Board

You can run one of the instance projects on the board and the other in one of the emulators. Follow these steps to run the client on the board and the server in one of the emulators:

> **Note:** Applications being run on an embedded platform, such as the Qualcomm Orion IoE or Raspberry Pi, must be signed. For more information, see the Oracle Java ME SDK 8 EA 2 *Getting Started Guide* for your embedded platform.

1. Right click on the first project (the server) and select Properties. In the Platform category select the device EmbaddedDevice1 (the emulator) and click OK. In the Application Description category set the value of the property Oracle-Demo-Network-Mode to **Server** and click OK.
2. Launch the first project (the server). It runs on the emulator and waits for a connection.
3. Right click on the second project (the client) and select Properties. In the Platform category select the device **EmbeddedExternalDevice1** (the board). In the Application Description category set the value of the property Oracle-Demo-Network-Mode to **Client** and click **OK**.

In the Application Descriptor category set the value of the property Oracle-Demo-Network-Address to the IP address of the computer where NetBeans 8.0 Beta is running and click OK.

4. Launch the second project (the client). It runs on the board and attempts to connect to the server. If successful, you see the following in the output tab of the first project (the server):

```
Connection accepted
Message received - Client messages
```

The TCP log of the board (the client) shows the following:

```
Connected to server 10.0.0.10 on port 5000
Message received - Server String
```

Running the PDAP Demo

This demo is a version of the PDAPDemo file browser.

Running the PDAPDemo on the Emulator

Follow these steps to run the demo on the `EmbeddedDevice1` emulator:

1. Create test files and directories inside the emulator's file system:

   `\`*username*`\javame-sdk\8.0_ea\work\EmbeddedDevice1\appdb\filesystem\root1`

2. Load the project in the Package window.
3. In the Platform category, select the device **EmbeddedDevice1** and click **OK**.
4. In the Device Selector window, right-click an **EmbeddedDevice1** emulator.
5. Select **Run Project** > **PDAPDemo** from the context menu.
6. Launch the project.
7. On the **EmbeddedDevice1** emulator, click the **Tools** menu and select **Manage File System** to see a list of mounted file systems.
8. Open a terminal emulator and create a Telnet connection to **localhost** on port **5001**.

   Note: The Telnet negotiation mode must be set to **Passive**. The negotiation mode can be set inside a Telnet client application (for example, PuTTY), by choosing **Category --> Connection --> Telnet --> Passive**.

9. A command line opens where you can browse the emulator's file system. You can use the following commands:
   - `cd` - change directory
   - `ls` - list information about the FILEs for the current directory
   - `new` - create new file or directory
   - `prop` - show properties of a file
   - `rm` - remove the file
   - `view` -View a file's content

Running PDAPDemo on the Reference Board

Follow these steps to run the demo on the reference board:

1. Right-click the project and select **Properties**. In the Platform category, select the device **EmbeddedExternalDevice1** and click **OK**.
2. Launch the project. It runs on the reference board.
3. Open a terminal emulator and create a raw connection to the IP address of the board on port **5001**.
4. The command line that opens is the same as the one you use when you run the demo on the emulator.

The file system of the demo is stored in the directory `java/`*IMletID* inside the SD card, where *IMletID* is a number that the AMS assigns to an IMlet during its installation.

Running the Light Tracker Demo

The Light Tracker demo is specifically aimed at showing off functionality on a embedded device, such as the Qualcomm Orion IoE reference platform.

In this demo, a certain number of LEDs are turned on and turned off on the board, in a sequence that you can control. It makes use of the Device Access API and the GPIO Port to demonstrate its functionality. It requires connection of an ADC channel to an on-board potentiometer.

For more information on the setup and behavior of the Light Tracker demo, see the LightTrackDemo `readme.txt` file, in the location where you have installed the Oracle Java ME SDK 8 EA 2 sample applications. For example:

```
C:\Java_ME_platform_SDK_8.0_EA\apps\LightTrackDemo
```

For more information on the Qualcomm Orion IoE platform, see the *Oracle Java ME Embedded Getting Started Guide for the Reference Platform (Qualcomm Orion IoE)*.

Running the System Controller Demo

The System Controller demo is specifically aimed at showing off functionality on a embedded device, such as the Qualcomm Orion IoE reference platform.

The purpose of this demo is to control the lifecycle of IMlets on the reference platform. It makes use of the following functionalities:

- Multitasking Virtual Machine (MVM)
- IMlet auto-start
- Application Management System (AMS) API
- Logging API
- General Purpose Input/Output (GPIO)
- Watchdog timer

For more information on the setup and behavior of the System Controller demo, see the SystemControllerDemo `readme.txt` file, in the location where you have installed the Oracle Java ME SDK 8 EA 2 sample applications. For example:

```
C:\Java_ME_platform_SDK_8.0_EA\apps\SystemControllerDemo
```

For more information on the Qualcomm Orion IoE platform, see the *Oracle Java ME Embedded Getting Started Guide for the Reference Platform (Qualcomm Orion IoE)*.

Troubleshooting

Sometimes a sample application does not run successfully. Often, the problem is your environment.

- Some demonstrations require specific setup and instructions. For example, if a sample uses web services and you are behind a firewall, you must configure the emulator's proxy server settings. See "Configuring the Web Browser and Proxy Settings."
- Because sample programs may be launched remotely, virus checking software can sometimes prevent them from running. In the console you see warnings that the emulator cannot connect.

  Consider configuring your antivirus software to allow access to sample application directories and components.

A

Using the Command Line Emulator

The Oracle Java ME SDK 8 EA 2 Embedded emulator can be started from a Windows command line. Once started, it runs and behaves the same as it does when started from NetBeans 8.0 Beta.

Starting the emulator from the Windows command line allows you to use a number of emulator options. For more information, see "Useful Emulator Commands".

Finding the Oracle Java ME SDK 8 EA 2 Emulator

You can find the Oracle Java ME SDK 8 EA 2 command line emulator in the `bin` directory of the Oracle Java ME SDK 8 EA 2 installation.

For example, if the Oracle Java ME SDK 8 EA 2 is installed in `C:\Java_ME_platform_SDK_8.0_EA`, then the emulator would be located at: `C:\Java_ME_platform_SDK_8.0_EA\bin\emulator.exe`

Using the Oracle Java ME SDK 8 EA 2 Emulator

To start the emulator from the Windows command line:

1. Open the Windows command line from the Start Menu, using **Start > Run**.
2. Change to the `bin` directory of the Oracle Java ME SDK 8 EA 2 distribution:

   C:\>**cd Java_ME_platform_SDK_8.0_EA\bin**
3. Enter the following command, as shown here:

   `emulator.exe -Xdevice:EmbeddedDevice1 -Xdescriptor:`*location_of_jad_file*

   For example:

   emulator.exe -Xdevice:EmbeddedDevice1 -Xdescriptor:C:\Java_ME_platform_SDK_8.0_EA\apps\sample\sample_imlet.jad

---

Note: You can run the `emulator` command without the `.exe` extension. It works both ways, with the extension and without

---

Useful Emulator Commands

This section provides a list of Oracle Java ME SDK 8 EA 2 emulator commands you may find useful, which can be entered on the Windows command line.

Note: For a complete list of Oracle Java ME SDK 8 EA 2 emulator commands, type: `emulator -help`.

- To show a list of installed IMlets, use the `-Xjam:list` subcommand:

  EmulatorDir`>emulator -Xjam:list`

- To see a list of all supported devices, use the `-Xquery` subcommand:

  EmulatorDir`>emulator -Xquery`

- To install a JAD over the air (OTA) and execute a IMlet, use the `-Xjam:install` subcommand:

  EmulatorDir`>emulator -Xjam:install=<JAD_file_URL>`

  For example:

  EmulatorDir`>emulator -Xjam:install=http://www.`*some_url*`.com/TestJAD.jad`

- To run an installed IMlet, use the `-Xjam:run` subcommand:

  EmulatorDir`>emulator -Xjam:run=[storage_name | storage_number]`

  Provide either the storage name or storage number for the IMlet to run. You can get the storage name and storage number from the list of IMlets shown by the `-Xjam:list` subcommand.

- To remove an installed IMlet, use the `-Xjam:remove` subcommand:

  EmulatorDir`>emulator -Xjam:remove=[storage_name | storage_number | all]`

  Provide either the storage name or storage number for the IMlet to remove. To remove all IMlets, use `all`. You can get the storage name and storage number from the list of IMlets shown by the `-Xjam:list` subcommand.

- To install a JAD file, execute the IMlet locally, and remove the IMlet when completed, use the `-Xdescriptor` subcommand:

  EmulatorDir`>emulator -Xdescriptor:<JAD_file_name>`

- To run in autotest mode, use the `-Xautotest` subcommand:

  EmulatorDir`>emulator -Xautotest:<JAD_file_URL>`

  For example:

  EmulatorDir`>emulator -Xautotest:http://127.0.0.1:8080/getNextApp.jad`

B

Installation and Runtime Security Guidelines

The Oracle Java ME SDK 8 EA 2 requires an execution model that makes certain network resources available for emulator execution. These required resources might include - but are not limited to - a variety of communication capabilities between product components.

> **Note:** the Oracle Java ME SDK 8 EA 2 installation and runtime system is fundamentally a developer system. It is not designed to guard against any malicious attacks from outside intruders.

During execution, the Oracle Java ME SDK 8 EA 2 architecture can present an insecure operating environment to the platform's installation file system, as well as its runtime environment. For this reason, it is critically important to observe the precautions outlined in these guidelines when installing and running the Oracle Java ME SDK 8 EA 2.

Maintaining Optimum Network Security

To maintain optimum network security, Oracle Java ME SDK 8 EA 2 can be installed and run in a "closed" network operating environment, where the Oracle Java ME SDK 8 EA 2 system is not connected directly to the Internet. Or, it can be connected to a secure company Intranet environment that can reduce unwanted exposure to malicious intrusion.

An example of an Oracle Java ME SDK 8 EA 2 requirement for an Internet connection is when wireless functionality requires a connection to the Internet to support communications with the wireless network infrastructure that is part of an Oracle Java ME SDK 8 EA 2 application execution process. Whether or not an Internet connection is required depends on the particular application running on Oracle Java ME SDK 8 EA 2. For example, some applications can use an HTTP connection.

In any case, if the Oracle Java ME SDK 8 EA 2 is open to any network access you must observe the following precautions to protect valuable resources from malicious intrusion:

- Installing the Oracle Java ME SDK 8 EA 2 Demos plugin is optional. Some sample projects use network access and open ports. Because the sample code does not include protection against malicious intrusion, you must ensure your environment is secure if you choose to install and run the sample projects.
- Install the Oracle Java ME SDK 8 EA 2 behind a secure firewall that strictly limits unauthorized network access to the Oracle Java ME SDK 8 EA 2 file system and services. Limit access privileges to those that are required for Oracle Java ME SDK

8 EA 2 usage while allowing all the bidirectional local network communications that are necessary for Oracle Java ME SDK 8 EA 2 functionality. The firewall configuration must support these requirements to run the Oracle Java ME SDK 8 EA 2 while also addressing them from a security standpoint.

- Follow the principle of "least privilege" by assigning the minimum set of system access permissions required for installation and execution of the Oracle Java ME SDK 8 EA 2.
- Do not store any data sensitive information on the same file system that is hosting the Oracle Java ME SDK 8 EA 2.
- To maintain the maximum level of security, make sure the operating system patches are up-to-date on the Oracle Java ME SDK 8 EA 2 host machine.

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 Wi-Fi or Bluetooth.

ADC

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

AMS

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

APDU

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

API

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

ARM

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

AT commands

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

AXF

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

BIP

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

CDMA

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

CLDC

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

Configuration

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

DAC

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

ETSI

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

GCF

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

GPIO

General Purpose Input/Output. Unassigned pins on an embedded platform that can be assigned or configured as needed by a developer.

GPIO Port

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

GSM

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

HTTP

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

HTTPS

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

ICCID

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

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

Similar to a MIDP 2.0 MIDlet, an IMlet is an small application specifically for running in an embedded environment. An IMlet uses classes defined by the MEEP 8.0 and CLDC 8.0 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. Each IMlet suite contains a Java application descriptor file (`.jad`), which lists the class names and files names for each IMlet, and a Java Archive file (`.jar`), which contains the class files and resource files for each IMlet

IMSI

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

I2C

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

ISA

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

JAD file

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

JAR file

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

JCP

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

JDTS

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

Java ME platform

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

JSR

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

Java Virtual Machine

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

KVM

A Java virtual machine designed to run in a small, limited memory device. The CLDC configuration was initially designed to run in a KVM.

MEEP

Oracle Java ME Embedded Profile. A profile for embedded (headless) devices, the MEEP specification (JSR 361) includes APIs for security, networking, connectivity, concurrency, and other functionality, and but not graphics and user interface APIs.

MSISDN

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

MVM

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

Obfuscation

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

Optional Package

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

Preemption

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

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.

RTX

The real-time operating system used on the Keil MCBSTM32F200 embedded platform.

SCWS

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

SD card

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

SIM

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

Slave Mode

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

Smart Card

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

SMS

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

SMSC

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

SOAP

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

SPI

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

SSL

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

SVM

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

Task

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

TCP/IP

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

Terminal Profile

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

UART

Universal Asynchronous Receiver/Transmitter. A piece of computer hardware that translates data between serial and parallel formats. It is used to facilitate communication between different kinds of peripheral devices, input/output streams, and embedded systems, to ensure universal communication between devices.

UICC

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

UMTS

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

URI

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

USAT

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

USB

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

USIM

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

WAE

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

WAP

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

Watchdog Timer

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

WCDMA

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

WMA

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

XML Schema

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

Index