Oracle® Java ME Embedded

Getting Started Guide for the Reference Platform (STM32429I-EVAL)

Release 8.1 Developer Preview

E61129-01

April 2015

This book describes how to install and run the Oracle Java ME Embedded software on the ST Micro STM32429I-EVAL reference platform.

Beta Draft

ORACLE

Oracle Java ME Embedded Getting Started Guide for the Reference Platform (STM32429I-EVAL), Release 8.1 Developer Preview

E61129-01

Contents

List of Figures

List of Tables

Preface

This book describes how to install Oracle Java ME Embedded software onto an ST Micro STM32429I-EVAL embedded device.

Audience

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

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

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

Operating System Commands

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

Shell Prompts

Shell	Prompt
Bourne shell and Korn shell	$

Shell	Prompt
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

Installing on the ST Micro STM32429I-EVAL Board

This chapter documents how to install the Java ME Embedded software onto an ST Micro STM32429I-EVAL board, including configuring the Java ME Embedded system, connecting to the command-line and logging interfaces, and performing basic configuration tasks.

The following items are required for developing on the ST Micro STM32429I-EVAL board:

- An ST Micro STM32429I-EVAL Embedded Board
- The Oracle Java ME Embedded Software Distribution for the ST Micro STM32429I-EVAL board, Version 8.1
- A desktop computer running Windows 7 or later with at least one USB port
- A 2GB or greater MicroSD card (with an SD adapter, if necessary, for connecting to the desktop computer)
- A USB-A to USB-B cable to flash the embedded board
- A networking LAN cable with RJ-45 interface, if you wish to communicate with the embedded board over a TCP/IP network
- A USB-A to MicroUSB cable, if you wish to communicate with the embedded board over a serial connection.
- A terminal emulation program, such as PuTTY

Downloading and Installing the MDK-ARM Lite Development Kit

To install the Oracle Java ME Embedded software on the reference board, first download and install version 5.12 or above of the MDK-ARM development kit. The MDK-ARM Core Version 5 development kit can be obtained from the following site:

`http://www2.keil.com/mdk5/install`

Once downloaded, install the MDK-ARM tool by double-clicking on the executable. Example projects do not need to be installed. If the application starts after installation, close it down.

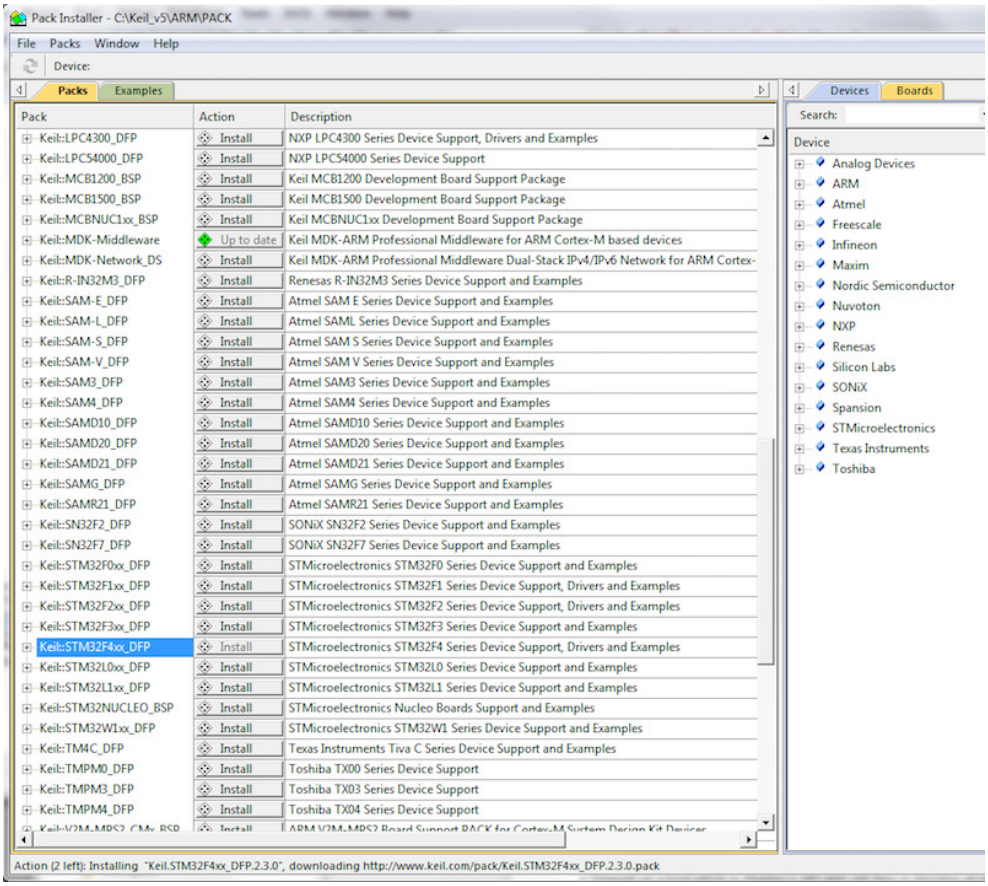
WARNING: The MDK-ARM Development Kit must be installed in a new directory, or previous versions of the software must be removed. Otherwise, the software may not be able to read older driver pack descriptions.

Next, install the ST-Link drivers to connect to the board. This is mandatory if the drivers are not already installed on your system. The ST-Link drivers can be obtained from the following site:

http://www.st.com/web/catalog/tools/FM147/SC1887/PF258168#

When the installation of both packages is complete, run the Keil uVision 5 application. Open the Pack Installer tool using the Project->Manage->Pack Installer... menu item, if it does not appear automatically, and install the Keil::STM32F4xx_DFP driver pack, as shown in Figure 1–1. Once this is completed, shut down the Keil uVision 5 application.

Figure 1–1 Driver Pack for the STM32429I-EVAL Embedded Board.



Downloading and Installing the PuTTY Terminal Emulator Program

Download the PuTTY Terminal Emulator Program (`putty.exe`) from the following site:

http://www.putty.org/

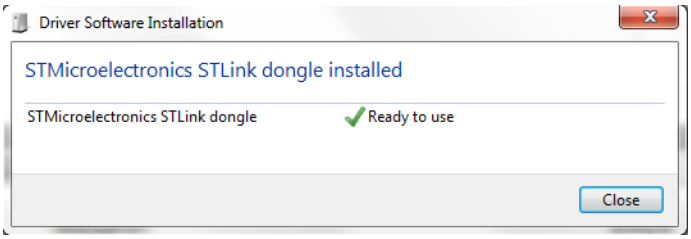
The terminal emulator executable is directly downloadable as `putty.exe`. The terminal emulator is used to connect to two separate sockets: one for the command-line interface (CLI) that issues commands to the board, and one for the logging or system output provided by the board.

Preparing the ST Micro STM32429I-EVAL Board

In order to use the ST Micro STM32429I-EVAL board for Java development, first prepare the board using the following steps:

- Set the JP12 jumper to the "PSU" state, which indicates that the board will be powered from the external adapter "PSU DC5V".
- Set the JP8 jumper to 3V3 position and ensure the LI-ION battery is inserted into the board's socket. Otherwise the system time and log setting will be reset every time the board is turned off.
- Connect the board to the computer using a USB-A to USB-B cable, connecting to the USB port next to the power connector.
- Connect an ethernet cable to the RJ-45 port on the board.
- Plug in the external power adapter delivered with the board to the "PSU DC5V" connector. The board should power up, and the LEDs and display should turn on. At this point, Windows will attempt to locate a compatible ST Link device driver. After searching the Windows Update drivers, it should eventually locate the version installed on the disk and present the dialog shown in Figure 1–2.

Figure 1–2 ST Link Dongle Driver Installation

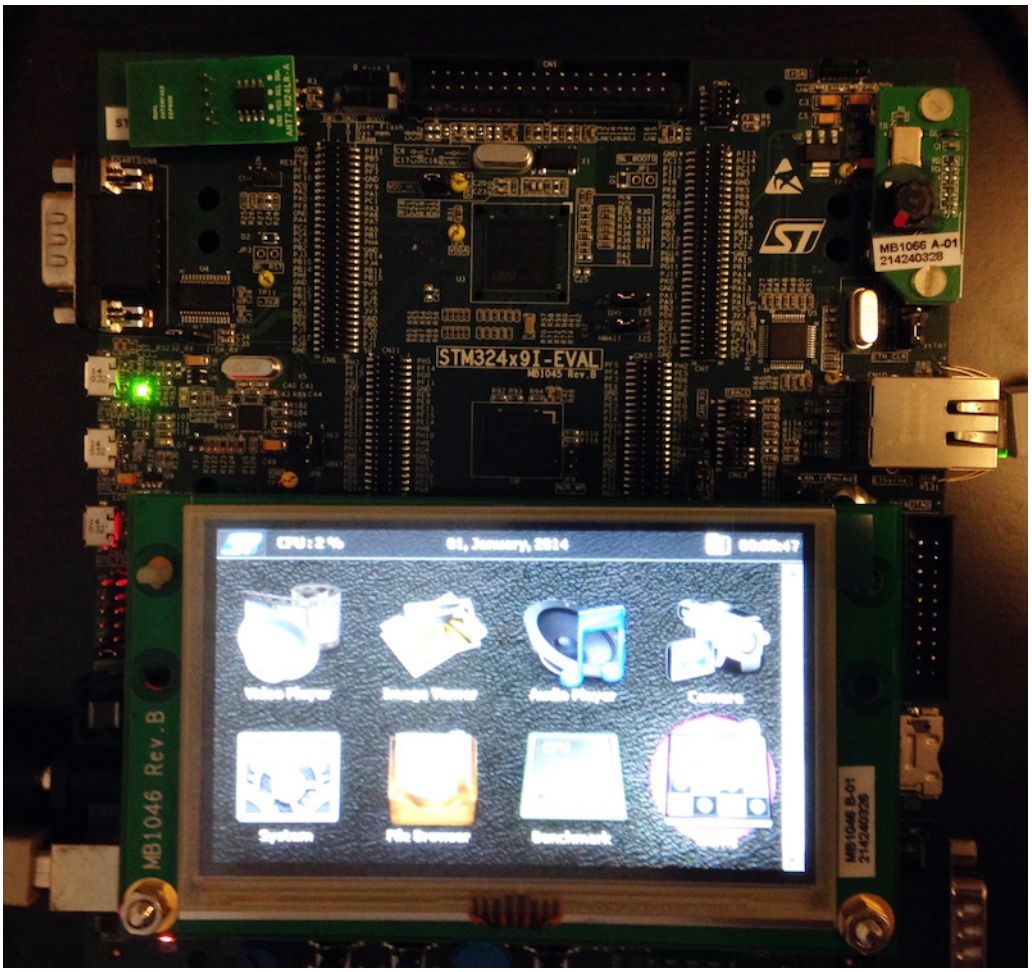


- Finally, ensure that the "STMicroelectronics STLink dongle" device appears in Windows Device Manager under Universal Serial Bus controllers or devices. The Device Manager can be accessed using the Windows Control Panel.

Connecting the ST Micro STM32429I-EVAL Board to the Computer

Ensure that the board is receiving power by verifying that the touchscreen is active. See Figure 1–3.

Figure 1–3 The ST Micro STM32429I-EVAL Board



When the board is first powered on, the touchscreeen may request a calibration from the user. It is not necessary to complete this calibration in order to use Java ME Embedded on the board.

Exploring the Oracle Java ME Distribution Bundle

At this point, unzip the Oracle Java ME Embedded Distribution 8.1 for the STM32429I-EVAL Embedded Board. The distribution bundle contains four directories that consist of the following important files:

/java/deploy.bat
The `deploy.bat` file is a Windows script file that invokes the uVision MDK-ARM tool to flash the board with the contents of the Oracle Java ME Emebedded binary file. You must edit this file to point to the installation directory of the Keil MDK-ARM tools directory (typically `C:\Keil_v5`) before flashing the embedded board.

/java/java.uvoptx, /java/java.uvprojx
These are uVision project files that are used by the Keil uVision MDK-ARM tool. You should not modify these files.

/java/jmee.axf
This is the Oracle Java ME Embedded binary file that will be flashed on the ST Micro board, stored in an ARM executable format. Do not modify this file.

/lib/classes.zip
This ZIP file contains the Java class structures for all publicly reachable classes used by the Java ME executable.

/sd_card/java/jwc_prop.ini
This is the main properties file for the Java ME binary executable. Modify this file before copying the `/sd_card` directory to the SD card that is inserted into the embedded board to control various runtime elements of the Oracle Java ME Embedded binary.

/sd_card/java/appdb/_main.ks
This the Java security keystore file that is used by the Java ME binary executable on the board. Do not attempt to modify this file directly. Instead, use the CLI keystore commands to access keys and certificates in this file. See Chapter 2 for more information on using the CLI proxy.

/sd_card/java/appdb/_policy.txt
This is the security policy file that defines policy groups for the Java ME binary executable on the board. Modify this file as needed to create your own security policies.

/sd_card/java/appdb/root
The root filesystem is a user-accessible directory that can be used to store and retrieve files and data through programs running on the board. Note that data above this directory is inaccessible by user programs.

/util/proxy.jar
This is an executable JAR file that can be used to connect to the CLI proxy to issue commands to the board. See Chapter 2 for more information on using the CLI proxy.

/util/AutoTestWrapper.jar
This is an executable JAR file that can be used to perform automatic testing when the Oracle Java ME Embedded binary is first booted. See the README in this directory for more information on performing automatic testing.

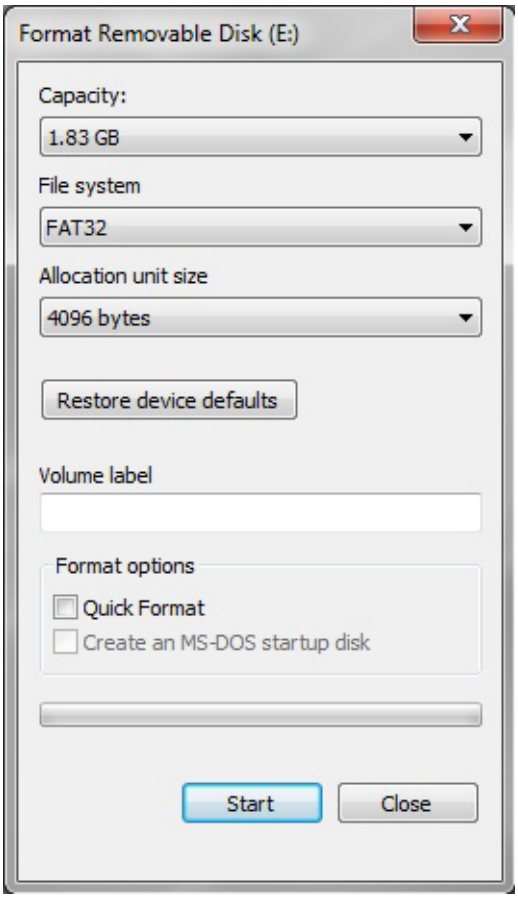
Setting Up the MicroSD Card

The MicroSD card contains configuration files used by the Java ME Embedded native distribution, including the initialization properties and security policy files. Ensure that you are using an SD card that is compatible with the embedded board.

Follow these steps to prepare the card:

1. Insert the card on the Windows desktop computer, select the card in the My Computer window, and right-click and select **Format...**
2. Select File System as FAT32, Allocation Unit Size as the default allocation unit size, and ensure that Quick Format is **not** selected, as shown in Figure 1–4. The Volume label is optional. Press the **Start** button. Once the formatting is completed, move to the next step.

Figure 1–4 Windows Formatter Settings



3. Copy the `java/` folder inside the `sd_card/` directory of the Oracle Java ME Embedded distribution to the root directory of the SD card. Do not eject the SD card yet.

Note that the Oracle Java ME Embedded runtime only supports 8.3 filenames on the SD card of the device.

Network Configuration

When the board boots, basic network initialization is performed using DHCP. However, if you wish to specify your own network configuration, you can modify the following parameters in the `java/jwc_prop.ini` file on the SD card.

Table 1–1 Ethernet Initialization Parameters

Parameter	Description
`system.network.ethernet.macaddress`	The MAC address that is assigned to the connected Ethernet device
`system.network.ethernet.ipmethod`	The method of obtaining an IP address for the configuration of the Ethernet interface, one of "`static`" or "`dhcp`". "`dhcp`" is used as the default.
`system.network.ethernet.ipaddress`	The IP address which is assigned to the device if a static IP method is chosen

Table 1–1 (Cont.) Ethernet Initialization Parameters

Parameter	Description
`system.network.ethernet.mask`	The IP address mask which is assigned to the Ethernet device if a static IP method is chosen
`system.network.ethernet.gateway`	The IP address of gateway which is assigned to Ethernet device if a static IP method is chosen. By default, the current IP address is used;
`system.network.dnsaddress`	The DNS server IP address which is used to resolve DNS requests during network activities, such as `Connector.open()` methods.
`system.network.secdnsaddress`	An alternative DNS server IP address used to resolve DNS requests during network activities, such as `Connector.open()` methods.

Setting the Runtime Clock

To set up the Runtime Clock (RTC) in STM32429I-EVAL board, the `rtc.time.value` property in the `java/jwc_prop.ini` file on the SD card must be set in the following format:

```
rtc.time.value = 2013/12/22 12:25:33 GMT
```

Note: Due to the persistent character of information storage in the RTC registers (including RTC settings), the board may occasionally hang during RTC initialization. To solve the problem, it is necessary to do the following when installing Java ME to the board:

1. Switch J8 jumper to the 3V3 position
2. Switch the STM32429I-EVAL board off
3. Disconnect all cables from the board (this is necessary as the board may take its power supply from the signal lines).
4. Wait 5 seconds
5. Reconnect all necessary cables and switch the board on again. Note that you may need to reset the time again.

Installing the Firmware on the Evaluation Board

The Java ME Embedded distribution contains a utility that uses the MDK-ARM software to erase and flash the Oracle Java ME Embedded binary onto the evaluation board. Use the following procedure to prepare the SD card and install the firmware:

1. Use the **Safely Remove Hardware and Eject Media** function on the Windows desktop to remove the SD card from the computer. (In Windows, for example, the Safely Remove Hardware and Eject Media tool is present in the lower right system tray near the clock.) Alternatively, you can select the SD card in the My Computer window, right click and select **Eject**. If necessary, remove the MicroSD card from the SD card housing.
2. Ensure that the board is not powered up by temporarily disconnecting the USB and power cables. Insert the MicroSD card into the SD card slot on the board.
3. Connect the board to the Windows desktop computer as you did previously, including the including the power cable, the network cable, and the USB-A to USB-B cable. The board should now be receiving power.

4. On the desktop, edit the file `deploy.bat` inside the directory `java/` in the Oracle Java ME Embedded software. Ensure that the `KEIL_SDK` variable contains the correct path to the MDK-ARM tool binary. For example:

   ```
   set KEIL_SDK=C:\Keil_v5
   ```

5. With the board connected, run the batch file `deploy.bat` to download the distribution to the evaluation board. The touchscreen display on the device should go blank, and the COM LED should quickly flash red and green.

6. Once completed, push the **Reset** button on the board to start the Java ME Embedded native platform. The runtime will then use the configuration files on the MicroSD card to initialize itself. If successful, the touchscreen on the device should look similar to Figure 1–5.

7. If desired, disconnect the USB-A to USB-B cable and plug a USB-A to MicroUSB cable into the OTG-HS connector, which is labeled as connector CN9 on the board, just below the large black DB9 connecter in Figure 1–5.

Figure 1–5 The ST Micro STM32429I-EVAL Touchscreen After Installing Java ME Embedded

Connecting to Logging Ports

Logging from the board can be enabled by setting the `log.method` setting in `jwc_prop.ini` file in the `/java` directory of the SD card. The possible values for this property are the default value of `NONE`, as well as `UART` and `ITM`. Table 1–2 gives a description of each of the values. Note that the `UART` and `ITM` values are mutually exclusive; only one method may be used at a time.

Table 1–2 Logging Methods for the STM32429I-EVAL

Log Method Value	Description
`NONE`	No logging information is produced
`UART`	Logging information is sent using the RS-232 connector marked CN8 on the board
`ITM`	Logging information is sent using the RS-232 ITM (Instrument Trace Macrocell).

The log method value is obtained using the following procedure:

- The Oracle Java ME Embedded runtime will first read the setting from backup SRAM powered from Vbat (if possible) and then initialize the logging with that value. Note that JP8 must be properly set to use Vbat, otherwise the settings will be lost. In the event of data loss, the Oracle Java ME Embedded runtime uses `ITM` for this stage as a default.
- The Oracle Java ME Embedded runtime will read the setting from the `jwc_prop.ini file` and compare it with the value loaded from SRAM. If the values do not match, or there is no saved setting, then the current value is changed to what was read from the `jwc_prop.ini` file. The SRAM value is then rewritten to this value in all cases.

If you wish to use UART logging, either connect an RS-232 cable or a USB-to-Serial adapter to the DB9 connector labeled CN8 on the STM32 board (which uses an RS-232 voltage level). Do **not** connect to the DB9 connected labeled CN22 on the board. Once the serial connection is made, use PuTTY to connect to the appropriate COM port using the following configuration:

- Speed 115200
- 8 bits
- No parity
- 1 stop bit

If you are using a USB-to-Serial adapter, you may be required to install a USB-to-COM driver that will map the USB port to a virtual COM port.

To use the `ITM` option, you will need either an external hardware debugger or the STM32 ST-LINK utility, which uses a Serial Wire Output (SWO) interface in conjunction with a built-in debugger.

Using the Display

The Oracle Java ME Embedded runtime can be used to display text only on the touchscreen display of the STM32429I-EVAL board. Access to the display can be obtained using the `javax.microedition.lui` package, with the following constraints:

- The `id` is "LCD Display"

- Vertical and horizontal scrolling are not supported
- Adjustments to the lighting are not supported
- The default background color is black
- The default text color is green
- The number of lines supported is 15
- The maximum number of characters per line is 52

2

Installing and Running Applications on the STM32429I-EVAL Board

Developers can run and debug IMlets on the ST Micro STM32429I-EVAL board directly from the NetBeans IDE 8.0.2, or by using the Oracle Java ME SDK 8.1. This chapter describes how to add the board to the Device Connections Manager of the Oracle Java ME SDK 8.1 as well as how to program an IMlet for the board.

Installing the Oracle Java ME SDK 8.1

Before developing for the ST Micro STM32429I-EVAL board, a modified version of the Oracle Java ME SDK 8.1 must be installed in two steps.

1. Install the Oracle Java ME SDK 8.1 distribution.
2. Install the Oracle Java ME STM Update Package version 8.1 using the Update Manager application of the Oracle Java ME SDK 8.1. You must install this update or the Java ME SDK tools will not be able to communicate with the ST Micro STM32429I-EVAL board using either a TCP/IP or serial connection.

Connecting to the ST Micro STM32429I-EVAL Embedded Board

There are two different ways to connect to the STM32429I-EVAL board.

1. Run NetBeans IDE 8.0.2, which will use the Oracle Java ME SDK. If you wish to use the NetBeans IDE, skip to the Section , "Using NetBeans with the STM32429I-EVAL Board" below.
2. Manually start a Developer Agent program on the desktop host and run commands using the Application Management System (AMS) CLI.

The Developer Agent program is a JAR file inside the `util` directory of the Oracle Java ME Embedded distribution, and is named `proxy.jar`. You can start the Developer Agent program on the desktop host computer either in a server or a client mode. After the Developer Agent program starts, you can use the AMS CLI to communicate with the board.

Server Mode Connection

The server mode is used by default. In this mode, start the Java runtime on the STM32429I-EVAL board. Then do the following.

1. Shut down the Oracle Java ME SDK Device Manager, if it is already running.
2. Change to the `util` directory on your desktop host and enter the following command. You should see an output similar to the following:

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

Client Mode Connection

To switch to a client mode connection, perform the following steps.

1. Edit the `jwc_prop.ini` file in the `bin` directory on the SD card of the board as follows:
   - Set the `proxy_connection_mode` property to the `client` value.
   - Set the `proxy.client_connection_address` property to the IP address of the host running the Developer Agent program.
2. Change to the `util` directory on your desktop host and enter the following command. You should see an output similar to the following:

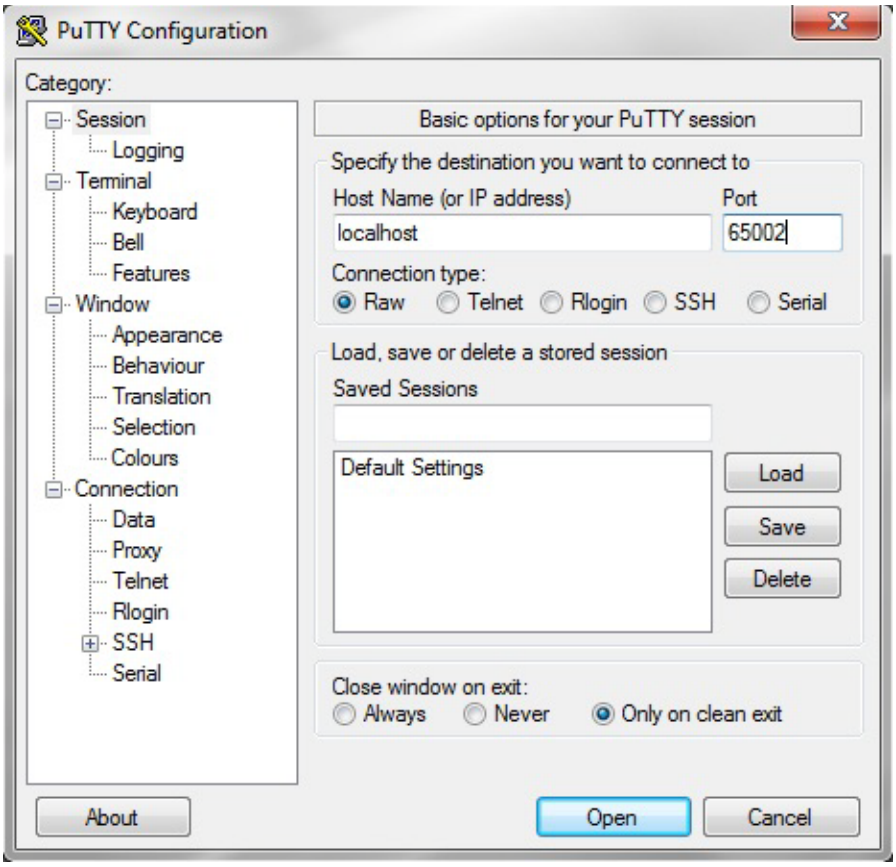
```
C:\mydir\util> java -jar proxy.jar
Starting with default parameters: -ServerSocketPort 2200 -jdbport 2801
Channel 8 CLOSED -> AVAILABLE
Waiting for device connections on port 2200
```

Running IMlets on STM32429I-EVAL Using the AMS CLI

The next step is to make a raw connection to the AMS CLI. Note, however, you must first run the Developer Agent program on the desktop host and the Java runtime on the board as described above, unless the Developer Agent was started automatically by Java ME SDK Device Manager.

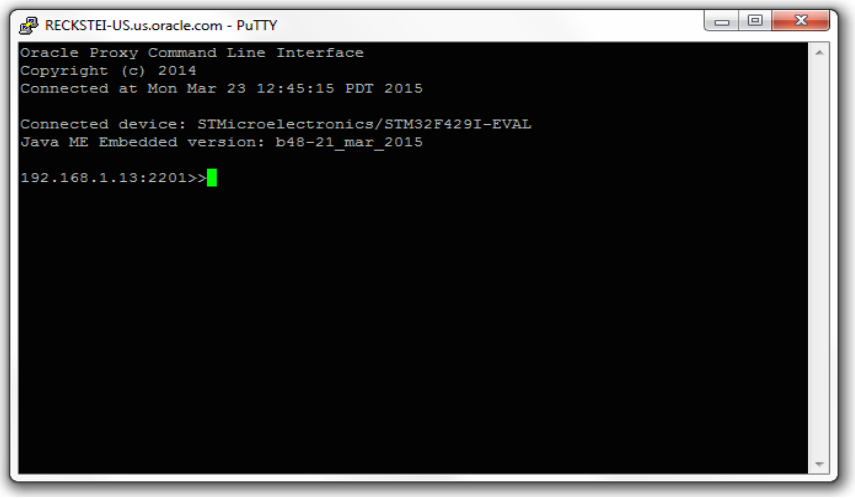
At this point, you can start a PuTTY executable file on your desktop computer. Use this to create Raw socket connections to the IP address of the host running the Developer Agent (typically localhost), and port 65002. For example, a connection to `localhost` and the port 65002 is shown in Figure 2–1. Be sure that the socket connection type is set to Raw and not SSH.

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



The window from port 65002 provides a CLI as shown in Figure 2–2.

Figure 2–2 Command-Line Interface to STM32429I-EVAL



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

You can use the command-line interface to run the AMS commands shown in Table 2–1. For a complete list of CLI commands, see the *Oracle Java ME Embedded Developer's Guide* at https://docs.oracle.com/javame/8.1/me-dev-guide/agentconsole.htm#sthref26

Table 2–1 AMS CLI Commands

Syntax	Description
`ams-info <INDEX>`	Show information about the installed IMlet. To determine the index of a specific IMlet, use the `ams-list` command.
`ams-install <URL> [auth=username:password][hostdownload]`	Install an IMlet using the specified JAR or JAD file, specified as a URL. An optional user name and password can be supplied for login information either in the URL or by using the `auth` parameter. When run without the `hostdownload` option, only `http://` URLs must be specified. The `hostdownload` option enables you to install an IMlet using the JAR file specified by the `file://` URL. Note that the JAR file must be located on the host.
`ams-list [INDEX or NAME|VENDOR]`	List all installed IMlet suites and their status or show the detail of a single suite.
`ams-remove <INDEX or NAME|VENDOR>`	Remove an installed IMlet.
`ams-run <INDEX or NAME|VENDOR> [IMLET_ID]`	Run the specified IMlet or the default if none is specified. You can specify optional debug parameter to run the IMlet in debug mode.
`ams-stop <INDEX or NAME|VENDOR> [IMLET_ID]`	Stop the specified IMlet or the default if none is specified.
`ams-update <INDEX or NAME|VENDOR>` [auth=username:password]	Update the installed IMlet. An optional user name and password can be supplied for login information by using the `auth` parameter.

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

```
192.168.1.12:2201>> ams-install file:///C:/some/directory/hello.jar hostdownload
<<ams-install,start install,file:///C:/some/directory/hello.jar
<<ams-install,install status: stage DONE, 0%
<<ams-install,install status: stage DONE, 100%
<<ams-install,OK,Install success

192.168.1.12:2201>> ams-install http://www.example.com/netdemo.jar hostdownload
<<ams-install,start install,http://www.example.com/netdemo.jar
<<ams-install,install status: stage DONE, 0%
<<ams-install,install status: stage DONE, 100%
<<ams-install,OK,Install success

192.168.1.12:2201>> ams-install http://www.example.com/notthere.jar hostdownload
<<ams-install,start install,http://www.example.com/notthere.jar
<<ams-install,install status: stage DONE, 100%
```

```
<<ams-install,FAIL,errorCode=JAR_NOT_FOUND, Invalid JAD Exception, reason: 20
(JAR_NOT_FOUND), message: Reason = 20, extra data:
http://www.example.com/notthere.jar
```

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

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

```
192.168.1.12:2201>> ams-list
<<ams-list,0.hello|Oracle,STOPPED
<<ams-list,1.netdemo|Oracle,STOPPED
<<ams-list,2.rs232dem|Oracle,RUNNING
<<ams-list,OK,3 suites are installed
```

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

```
192.168.1.12:2201>> ams-remove 0
<<ams-remove,OK,removed
```

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

```
192.168.1.12:2201>> ams-list
<<ams-list,1.netdemo|Oracle,STOPPED
<<ams-list,2.rs232dem|Oracle,RUNNING
<<ams-list,OK,2 suites are installed
```

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

```
192.168.1.12:2201>> ams-run 1
<<ams-run,OK,started

192.168.1.12:2201>> ams-list
<<ams-list,1.netdemo|Oracle,RUNNING
<<ams-list,2.rs232dem|Oracle,RUNNING
<<ams-list,OK,2 suites are installed
```

You can use the commands summarized in Table 2–2 to modify system properties.

Table 2–2 Security and Properties Commands

Syntax	Description
`help [command name]`	List the available commands or detailed usages for a single command.
`properties-list` [-l]	Show the list of names of properties which control the Java ME runtime (properties that are set in the `jwc_prop.ini` file). Use the -l parameter to show a long-listing format including the property types, values, and readonly flags.
`get-property <NAME> [-i]`	Return a value of the property identified by `<NAME>`.
`set-property <NAME> <VALUE>`	Set a property identified by `<NAME>` with the value `<VALUE>`.
`save-properties`	Save properties to an internal storage.
`blacklist -client <NAME>`	Blacklist clients and applications.
`blacklist -app <NAME|VENDOR>`	

File system commands are summarized in Table 2–3. Note that the system can only access files inside of the `/java/appdb` directory, and cannot use relative paths that shift the current location outside of this directory (for example, using ". ." in the path).

Table 2–3 File System Commands

Syntax	Description
`cd <deviceDirectoryName>`	Change the working directory on the device.
`delete <deviceFileName>`	Delete a file on the device.
`get <deviceFileName> <hostFileName>`	Copy a file from the device to the host.
`ls [<deviceDirectoryName>]`	Display a list of files and subdirectories in a device directory. In a result listing, subdirectories are marked with a forward slash.
`mkdir <deviceDirectoryName>`	Create a directory on the device.
`pwd`	Write the current working directory on the device.
`put <hostFileName> <deviceFileName>`	Copy a local host file to the device.
`rmdir <deviceDirectoryName>`	Deletes an empty directory on the device.

Network commands are shown in Table 2–4.

Table 2–4 Networking Commands

Syntax	Description
`net-info`	Show the networking information of the system.
`net-reconnect`	Reconnects the network and reboots Java.

The CLI supports working with multiple devices. You can use the device commands summarized in Table 2–5.

Table 2–5 Device Commands

Syntax	Description
`device-list`	List all connected devices.
`device-change <INDEX>`	Make the specified device current.
`shutdown [-r]`	Perform either a shutdown of the board or a reboot if the `-r` parameter has been passed.
`exit`	Terminate the current CLI session.

You can use the keystore commands summarized in Table 2–6.

Table 2–6 Keystore Commands

Syntax	Description
`ks-delete (-owner <owner name> | -number <key number>)`	Delete a key from a ME store.

Table 2–6 (Cont.) Keystore Commands

Syntax	Description
`ks-import [-keystore <filename>] [-storepass <password>] [-keypass <password>] [-alias <key alias>]`	Import a public key from a JCE keystore into a ME keystore.
`ks-list`	List the owner and validity period of each key in a ME keystore.

Using NetBeans with the STM32429I-EVAL Board

Running and debugging IMlet projects on the ST Micro STM32429I-EVAL board using the NetBeans IDE 8.0.2 requires the following software:

- NetBeans IDE 8.0.2
- Oracle Java ME SDK 8.1 with STM32429I-EVAL Update
- Oracle Java ME SDK 8.1 plugins for NetBeans

To install the Java ME SDK 8.1 toolkit in NetBeans 8.0.2, follow these instructions:

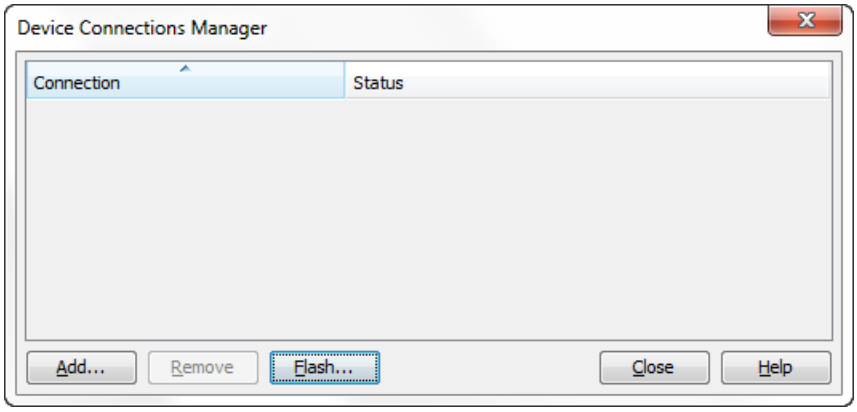
1. Select Tools->Plugins, and raise the Plugins dialog. Select the Installed tab, and activate the Java ME plugin.
2. Extract the Java ME NetBeans plugins from the ZIP file, placing them in their own directory.
3. Select the Downloaded tab in the Plugins dialog, and press the Add Plugins... button. Browse to the directory containing the NetBeans Java ME plugins, and select all the files with a `.nbm` extension, press the Open button to choose them, then press the Install button to install each of the plugins. You can safely ignore any warnings about the updates not being digitally signed. After the plugins are installed, restart the IDE.

Adding the STM32429I-EVAL Board to the Device Connection Manager

Follow these steps to add the STM32429I-EVAL board to the Device Connections Manager in Oracle Java ME SDK 8.1:

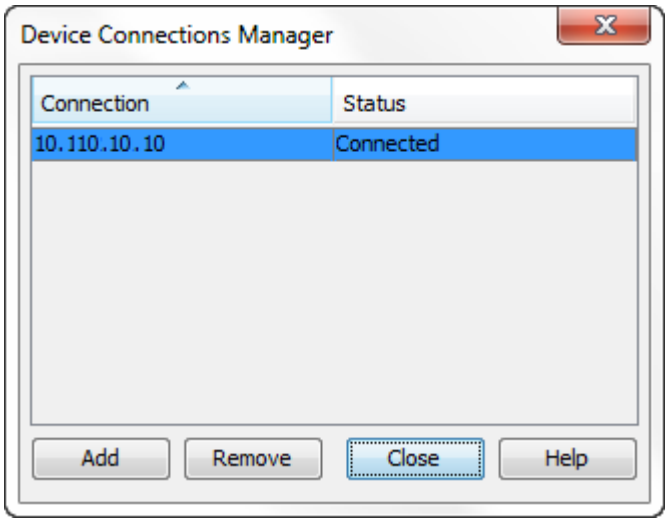
1. Ensure that the Developer Agent program is not running on the desktop computer.
2. Start the Oracle Java ME SDK 8.1 Device Connections Manager (located at <*SDK Installation Folder*>`/bin/device-manager.exe`) and click its Device Manager icon in the taskbar. A Device Connections Manager window is shown in Figure 2–3.

Figure 2–3 Device Connections Manager Window



3. Click the **Add** button, ensure that the IP Address or Host Name list contains the correct IP address of the ST Micro STM32429I-EVAL board, and click **OK**. You can verify the current IP address of the board by checking the touchscreen on the STM32429I-EVAL embedded board.
4. After the STM32429I-EVAL board is registered, its IP address is listed on the Device Connections Manager list and its status is Connected as shown in Figure 2–4.

Figure 2–4 Device Connections Manager Window with STM32429I-EVAL Connected



Assigning the STM32429I-EVAL Board to Your Project

There are two ways to assign the STM32429I-EVAL board to your project:

- Using an existing NetBeans Project with an IMlet you want to run or debug.
- Creating a new NetBeans project.

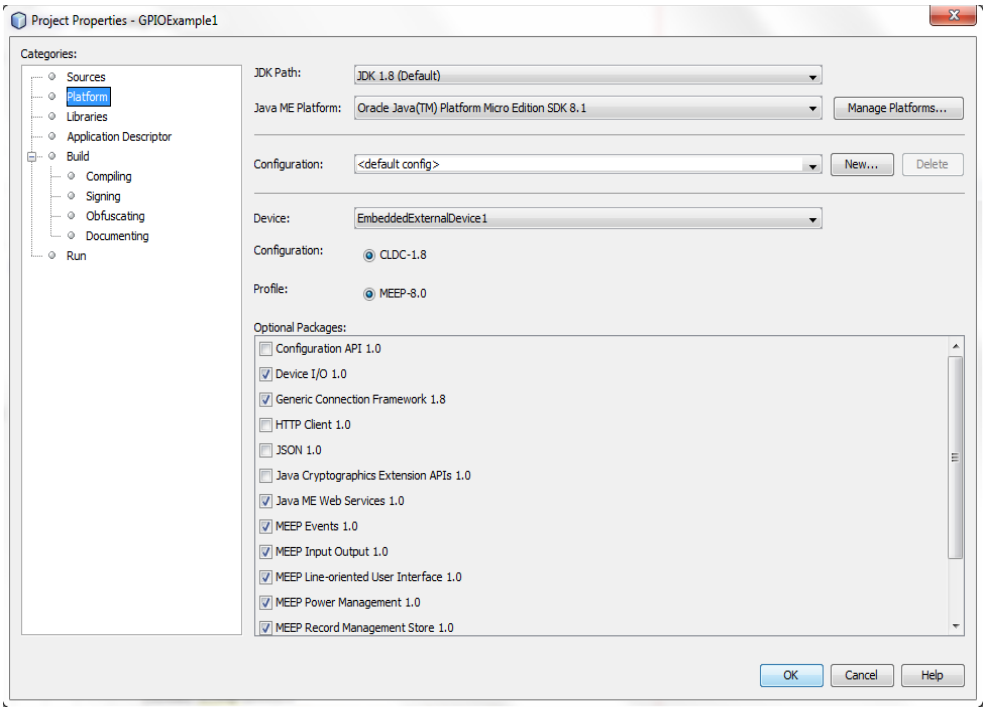
After you assign the board to your project, clicking **Run Project** in the NetBeans IDE runs your IMlets on the board instead of on the emulator.

Using an Existing NetBeans Project

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

1. Right-click your project and select **Properties**.
2. Select the **Platform** category on the properties window.
3. Select the Platform Type (CLDC) and ensure that **Oracle Java(TM) Platform Micro Edition SDK 8.1** is selected in the Java ME Platform list.
4. Select **EmbeddedExternalDevice1** from the Device drop-down list, as shown in Figure 2–5. Select (or deselect) from the list of Optional Packages as needed for your project, and click **OK**.

Figure 2–5 Adding a Device to Your Project

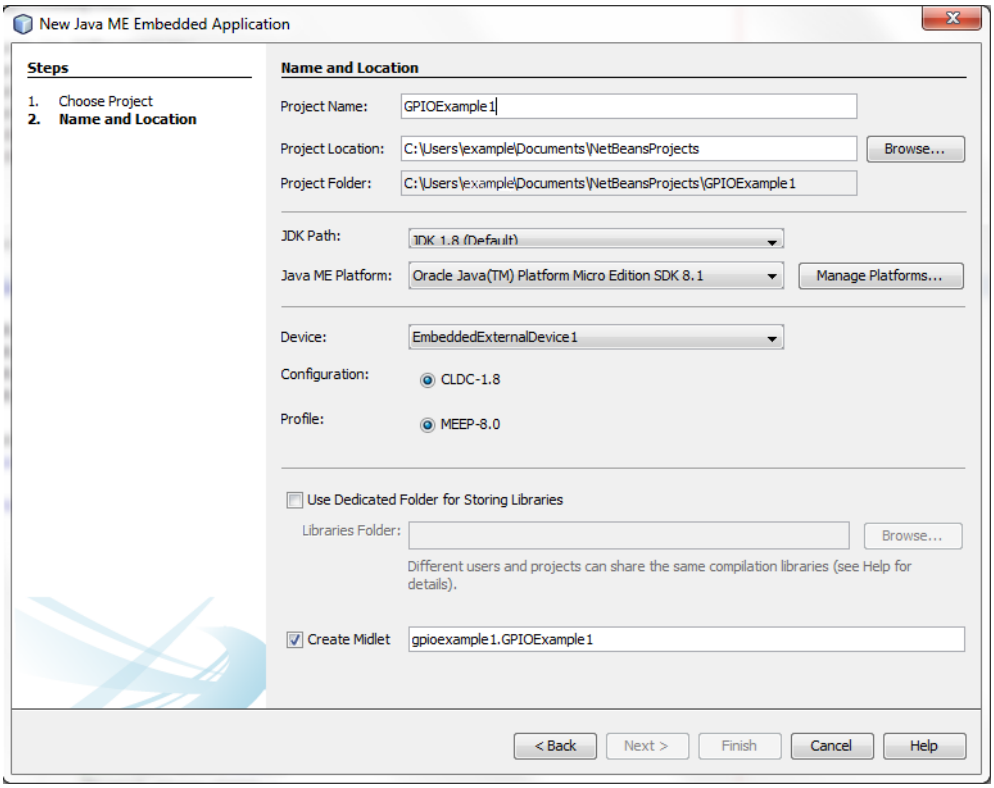


Creating a New NetBeans Project

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

1. Select **File** then **New Project**.
2. Select the **Java ME Embedded** category and **Java ME Embedded Application** in Projects pane. Click **Next**.
3. Provide a project name, for example, `ME8EmbeddedApplication1`. Ensure that the Java ME Platform is **Oracle Java(TM) Platform Micro Edition SDK 8.1** and the **Create Midlet** option is selected.
4. Select **EmbeddedExternalDevice1** from the Device drop-down list and click **Finish**, as shown in Figure 2–6.

Figure 2–6 Creating a New Project



When the new project is created, it is displayed in NetBeans IDE with the name `GPIOExample1`.

Sample Source Code

Now you can update the generic project that you created with the sample code shown in the following example. This sample application obtains an object for GPIO pins 1 through 4 (representing four LEDs on the board) from the `DeviceManager` object, and repeatedly toggles their high/low values.

```
package gpioexample1;

import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.microedition.midlet.MIDlet;
import jdk.dio.DeviceManager;
import jdk.dio.gpio.GPIOPin;

public class GPIOExample1 extends MIDlet {

    GPIOPin pin1;
    GPIOPin pin2;
    GPIOPin pin3;
    GPIOPin pin4;

    @Override
    public void startApp() {

        try {
```

```
            pin1 = (GPIOPin) DeviceManager.open(1);
            pin2 = (GPIOPin) DeviceManager.open(2);
            pin3 = (GPIOPin) DeviceManager.open(3);
            pin4 = (GPIOPin) DeviceManager.open(4);

            for (int i = 0; i < 20; i++) {
                System.out.println("Setting pins to true...");
                pin1.setValue(true);
                pin2.setValue(true);
                pin3.setValue(true);
                pin4.setValue(true);
                Thread.sleep(2000);
                System.out.println("Setting pins to false...");
                pin1.setValue(false);
                pin2.setValue(false);
                pin3.setValue(false);
                pin4.setValue(false);
                Thread.sleep(2000);
            }
        } catch (IOException | InterruptedException ex) {
       Logger.getLogger(GPIOExample1.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    @Override
    public void pauseApp() {
    }

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

In the NetBeans Projects window, you see the sample project with the file `GPIOExample1.java`. Follow these steps:

1. Double-click the `GPIOExample1.java` file in the Projects window.
2. Copy the sample code and paste it in the Source window.
3. Clean and build the `GPIOExample1` project by clicking on the hammer-and-broom icon in the NetBeans toolbar or by selecting **Run** then **Clean and Build Project (GPIOExample1)**.
4. Before running the example, we must add the appropriate permissions for the Device I/O APIs to access the peripherals.

Accessing Peripherals

Applications such as the previous example that require access to Device I/O APIs must request the appropriate permissions. For more information about the various permissions required to access the Device I/O APIs, see the *Device I/O API 1.0* specification at:

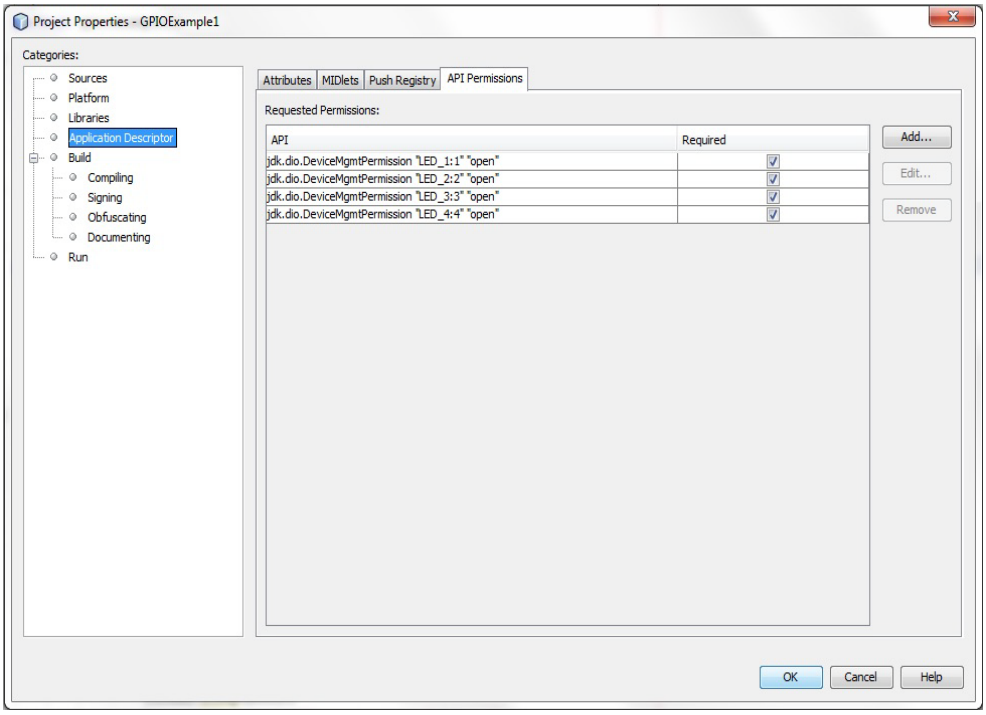
http://docs.oracle.com/javame/8.0/api/dio/api/index.html

Adding API Permissions

Follow these steps to add the necessary API permissions for the previous example:

1. In **NetBeans**, right-click the project name (**GPIOExample1** in this example) and select **Properties**.
2. Click **Application Descriptor**, then in the resulting pane, click **API Permissions**.
3. Click the **Add** button, and add the `jdk.dio.DeviceMgmtPermission` API, as shown in Figure 2–7.
4. Click **OK** to close the project properties dialog.

Figure 2–7 Adding API Permissions with NetBeans



5. If you are not using an IDE, you can manually modify the application descriptor file to contain the following permissions:

```
MIDlet-Permission-1: jdk.dio.DeviceMgmtPermission "LED_1:1" "open"
MIDlet-Permission-1: jdk.dio.DeviceMgmtPermission "LED_2:2" "open"
MIDlet-Permission-1: jdk.dio.DeviceMgmtPermission "LED_3:3" "open"
MIDlet-Permission-1: jdk.dio.DeviceMgmtPermission "LED_4:4" "open"
```

6. Compile and run the application.

Signing Applications

In a production environment, applications must be digitally signed in order to access peripherals. The use of digital signatures requires trusted certificates that are shared between both the NetBeans IDE and the embedded device.

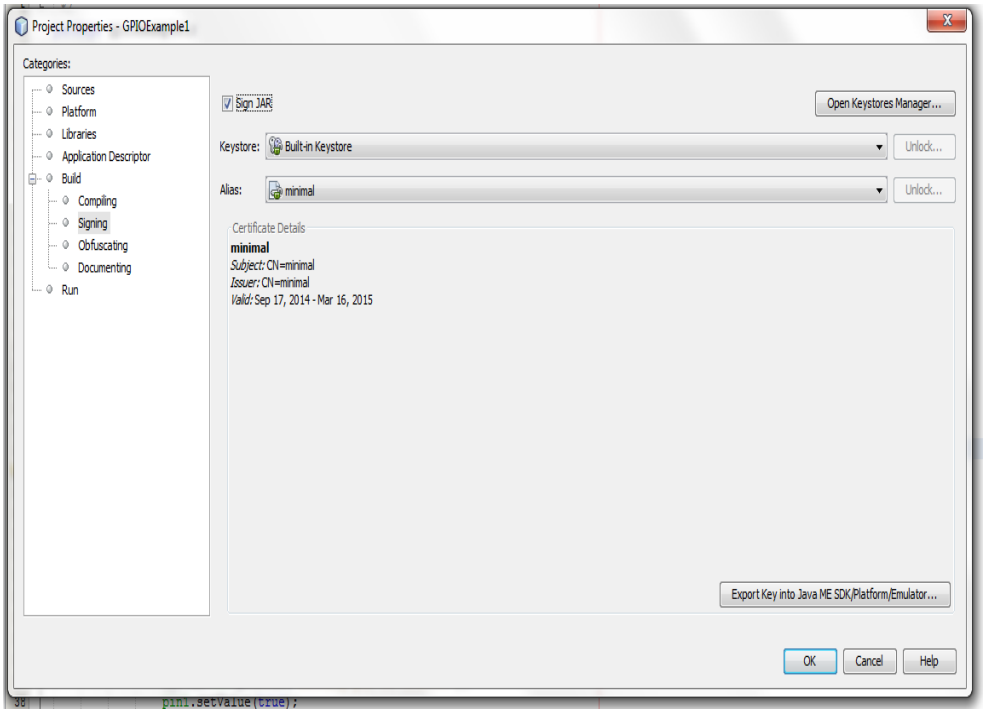
Signing Application Using the NetBeans IDE

The NetBeans IDE enables developers both to sign the applications with a local certificate and then upload that certificate to the device. Follow these steps:

1. Right-click the project name and select **Properties**.
2. Under the **Build** category, click **Signing**.

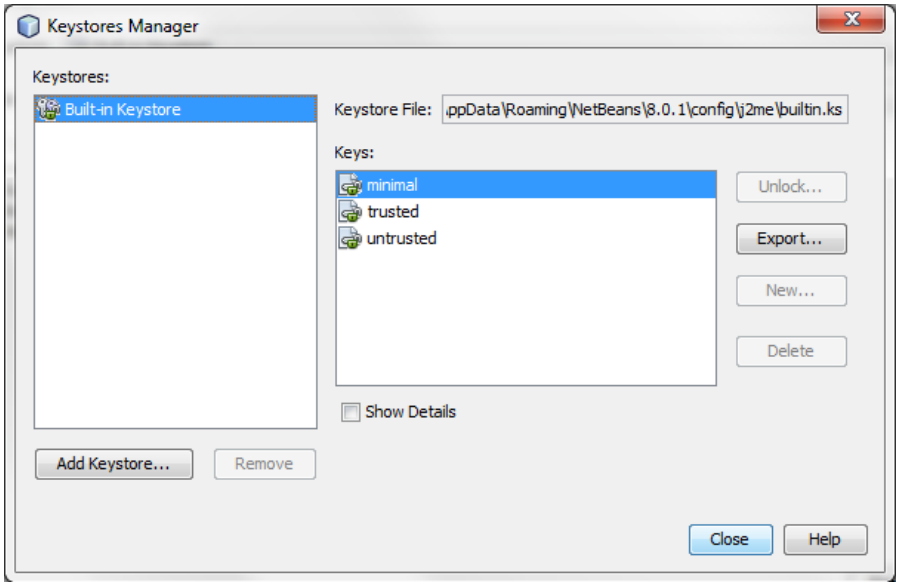
3. Select **Sign JAR** and specify a certificate to sign with as shown in Figure 2–8.

Figure 2–8 Signing Application JAR with NetBeans



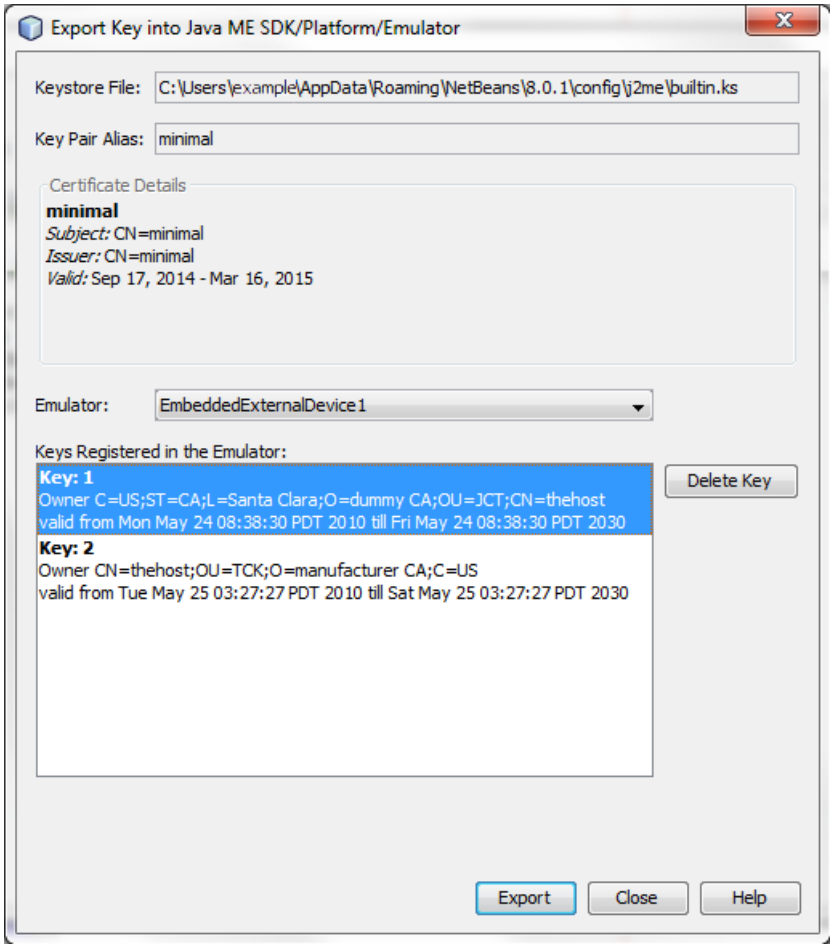
4. Click **Open Keystores Manager**.
5. Select the key and click **Export** as shown in Figure 2–9.

Figure 2–9 Keystores Manager Window



6. In the Export Key window, select the **EmbeddedExternalDevice1** in the Emulator drop-down list, and click **Export** as shown in Figure 2–10.

Figure 2–10 Exporting Key on a Device



7. Edit the `_policy.txt` file from the `/appdb` directory of the SD card on the board and add a section with the client name and a set of permissions. For more information about the policy file format, see the External Client Policy Format section in the *Java ME Embedded Profile 8* specification.
8. Ensure that the certificate with the specified common name (CN) is associated with the client by creating a section with a header that is identical to the certificate that was just uploaded to the board. (Remove the `untrusted` section once it is working). For example:

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

9. Copy the modified `_policy.txt` file back to the `/appdb` directory on the SD card.

Method #2: Signing an Application Using the Command Line

If you are not using the NetBeans IDE, then you can sign your IMlet using the command line. Follow the instructions on how to set up a keystore with a local certificate that can be used to sign the applications:

1. Generate a new self-signed certificate with the following command on the desktop, using the `keytool` that is shipped with the Oracle Java SE JDK.

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

This command generates a 2048-bit RSA key pair and a self-signed certificate, placing them in a new keystore with a keystore password of `spass` and a key password of `kpass` that is valid for 360 days. You can change both passwords as desired.

2. Perform the following command using the `mekeytool.exe` command (or alternatively `java -jar MEKeyTool.jar...` if your distribution contains only that) that ships with the Oracle Java ME SDK 8.1 distribution. Note that the board must be connected and recognized using the Oracle Java ME Device Manager. In this example, it is referred to as `ExternalEmbeddedDevice1`.

   ```
   {mekeytool} -import -keystore keystore.ks
   -Xdevice:ExternalEmbeddedDevice1
   ```

   This command imports the information in `mykeystore.ks` that you just created to the device currently connected as `ExternalEmbeddedDevice1`. See Appendix D in the Java ME Embedded Developers Guide for more information on using the `mekeytool`.

   Alternatively, the user can use the `ks-import` tool on the CLI to import information into the device keystore.

3. Modify the `_policy.txt` file on the SD card so that the AMS on the embedded board will accept the certificate, as in the previous method.

4. Use the `jadtool` command to sign your application before deploying it to the STM32429I-EVAL board. If you would like more information about the `jadtool` command, see Appendix D in the Java ME Embedded Developers Guide.

   ```
   jadtool -addcert -chainnum 1 -alias myalias -keystore mykeystore.ks
   -storepass spass -inputkad myjad.jad -outputjad myjad.jad
   ```

   ```
   jadtool -addjarsig -chainnum 1 -jarfile myjar.jar -alias myalias
   -keystore mykeystore.ks -storepass spass -keypass kpass -inputjad
   myjad.jad -outputjad myjad.jad
   ```

3

Troubleshooting

This chapter contains a list of common problems that you may encounter while installing and running the Oracle Java ME Embedded software on the ST Micro STM32429I-EVAL board. This chapter provides information on the causes of these problems and possible solutions for them.

The common problems in this chapter are grouped in four categories:

- Installing the Firmware on the Board
- Working with Oracle Java ME Embedded on the Board
- Using the Board with the Oracle Java ME SDK and the NetBeans IDE
- Development Log

Installing the Firmware on the Board

Table 3–1 contains information about problems and solutions when installing the firmware on the board.

Table 3–1 Problems and Solutions - Installing the Firmware on the Board

Problem	Cause	Solution
The `deploy.bat` script fails and the firmware is not installed on the board.	The correct drivers are not installed into the MDK-ARM software for the board.	Ensure that the correct drivers for the board are downloaded in the MDK-ARM tool.
MDK-ARM states "The following Device Family Pack(s) are required by the project: Keil STM32F4xx_DFP:2.3.0"	The Device Pack drivers for the ST Micro board have not been downloaded and installed into MDK-ARM.	Use the Pack Installer tool in MDK-ARM to download and install the proper drivers. See Chapter 1 for more details.
Pop-up windows at the start of deployment occurs with message "Invalid ROM Table"	Problem with Keil MDK-ARM tool.	See `http://www.keil.com/support/docs/3566.htm` for more information.

Working with Oracle Java ME Embedded on the Board

Table 3–2 contains information about problems and solutions when starting the runtime on the board.

Table 3–2 Problems and Solutions - Starting Oracle Java ME Embedded on the Board

Problem	Cause	Solution
Oracle Java ME Embedded fails to start on the board, even after the firmware is installed correctly.	The SD card is not inserted correctly.	Eject the SD card from the board, insert it again and press the Reset button.
(*continued*)	The SD card is not formatted correctly using FAT32.	Format the SD card using the Windows format tool and copy the files from the Oracle Java ME Embedded distribution. See the section Setting Up the MicroSD Card in Chapter 1. Insert the SD card on the board and press the Reset button.
(*continued*)	Some files in the SD card are marked as read only.	Change the attributes of all files in the SD card and assign them write permissions. Insert the SD card on the board and press the Reset button.
(*continued*)	In rare cases, the file system of the SD card may be corrupted.	Re-format and re-install the contents of the SD card. See the section Setting Up the MicroSD Card in Chapter 1. Insert the SD card on the board and press the Reset button.
Oracle Java ME Embedded fails to properly set the clock on the board.	The `jwc_prop.ini` configuration file was not configured prior to board initialization.	Configure the date and time in the `jwc_prop.ini` file on the SD card and reset the board.
(continued)	The JP8 jumper on the board is set to VBAT, not 3V3	Set the jumper to 3V3.

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

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

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

Problem	Cause	Solution
The Java ME menu does not appear; unable to find the Device Selector.	The Oracle Java ME SDK plugin for the IDE is not installed.	Install the Oracle Java ME SDK plugin that is specifically for your development IDE, as described in Chapter 2.
The runtime on the board has problems accessing files or it is unstable.	The SD card is not supported or it is not formatted correctly.	Ensure that you are using a supported SD card and that you format it using the Windows formatting tool. See the section Setting Up the MicroSD Card in Chapter 1. Insert the SD card on the board and press the Reset button.
(*continued*)	Thunderbird is using a port that is needed for communication with the board.	Close `thunderbird.exe` during the debugging session.
An authorization failure is given when an IMlet attempts to install or access any Device Access API.	The IMlet is not signed.	Sign the IMlet using a keystore with a trusted certificate authority (CA), or run the IMlet in the untrusted security zone.
(*continued*)	The proper Device I/O API permissions have not been requested.	See Chapter 2 for information on how to add API permissions to a project.

Table 3–3 (Cont.) Problems and Solutions - Oracle Java ME SDK and the NetBeans IDE

Problem	Cause	Solution
(continued)	The date on the board may invalidate the certificate used to authenticate the digital signature.	See Chapter 1 for the procedure on how to reset the RTC clock on the board.
Only four commands are available in the AMS CLI after connecting to the board.	The CLI proxy did not successfully connect to the board.	Ensure that an instance of the CLI proxy is not already running. Shut down any instances of the Oracle Java ME SDK Device Manager, which contains its own proxy and may already be using that port.
IMLet installation fails with a `JAD_NOT_FOUND` error	Firewall on a host which is sharing a JAD and JAR files is blocking an incoming HTTP or other requests.	Disable the firewall.
(continued)	Proxy issue.	Ensure proxy is disabled in Java Control panel and in Windows System Network settings

Development Log

The development log of the Oracle Java ME Embedded software can help you diagnose problems that arise when running IMlets on the board. The development log is covered in the section Connecting to Logging Ports in Chapter 1.

1. Connect a serial cable from the computer to the board. Use the CN8 connector.
2. Ensure that the `log.method` property from the `jwc_prop.ini` file contains the value `UART` in its list.
3. Open a terminal emulator on the computer, such as PuTTY.
4. Choose a serial connection and set the following options:
   - Speed: 115200
   - Data bits: 8
   - Stop bits: 1
   - Parity: None
   - Flow control: XON/XOFF

   In PuTTY these options are in the category **Connection** > **Serial**.
5. Open the connection. The system log appears on the terminal.

Output Stream Buffering

For optimization reasons, this implementation of the Oracle Java ME Embedded runtime does not provide `OutputStreamWriter` buffering. As such, the network layer on this platform does not provide `OutputStream` buffering for sockets. Consequently, any applications that are writing strings to a socket will have poor performance.

To solve this problem, simply wrap a socket's `OutputStream` with an additional `ByteArrayOutputStream`, as shown in this example:

```
final OutputStream socketOutputStream = socketConnection.openOutputStream();

// Socket performance workaround:
```

```
// make additional buffering around socket's output stream

OutputStream bufferedOutputStream = new ByteArrayOutputStream() {
    public void flush() {
        try {
            writeTo(socketOutputStream);
            reset();
            socketOutputStream.flush();
        } catch (IOException ioe) {}
    }
};

Writer writer = new OutputStreamWriter(bufferedOutputStream);
```

A

ST Micro STM32429I-EVAL Board Peripheral List

This appendix describes the proper ID and names for the various peripheral ports and buses for the ST Micro STM32429I-EVAL embedded board, which are accessible using the Device Access APIs. Note that any IMlet that accesses the Device Access APIs must be digitally signed using a trusted certificate authority. An IMlet that is not signed will encounter an authentication error when attempting to access the Device Access APIs.

Analog-to-Digital Converter (ADC)

The following ADC channels are pre-configured.

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
3000	Potentiometer	PF10	controllerNumber = 2 channelNumber = 8 resolution = 12 samplingInterval = 46857 samplingTime = 46857

Additional notes:

- Potentiometer (RV1) is connected to ADC3 channel 8 via GPIO PF10
- For the sampling interval, only a limited set of values are supported: 11810, 14857, or 46857 usec. Setting the sampling interval to any other value will force it to be reset to the closest legal value.
- After a call to `ADCChannel.startMonitoring()`, a `MonitoringListener` cannot be called more than one time around the boundary because of jitter.
- Note the following conditions on `ADCChannelConfig` parameters: the resolution is a 12-bit fixed value regardless of the input parameter, and the sampling interval and the sampling time are the same internally. `PERIPHERAL_CONFIG_DEFAULT` is supported, and the default value is 46857 usec.

GPIO Pins

The following GPIO pins are pre-configured:

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
1	LED_1	Green LED (R192)	controllerNumber = 6 pinNumber = 6 direction = GPIOPinConfig.DIR_OUTPUT_ONLY mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL trigger = GPIOPinConfig.TRIGGER_NONE initValue = false
2	LED_2	Orange LED (R191)	controllerNumber = 6 pinNumber = 7 direction = GPIOPinConfig.DIR_OUTPUT_ONLY mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL trigger = GPIOPinConfig.TRIGGER_NONE initValue = false
3	LED_3	Red LED (R190)	controllerNumber = 6 pinNumber = 10 direction = GPIOPinConfig.DIR_OUTPUT_ONLY mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL trigger = GPIOPinConfig.TRIGGER_NONE initValue = false
4	LED 4	Blue LED (R189)	controllerNumber = 6 pinNumber = 12 direction = GPIOPinConfig.DIR_OUTPUT_ONLY mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL trigger = GPIOPinConfig.TRIGGER_NONE initValue = false
5	BUTTON 1	Wake up Button (B2)	controllerNumber = 0 pinNumber = 0 direction = GPIOPinConfig.DIR_INPUT_ONLY mode = GPIOPinConfig.MODE_INPUT_PULL_DOWN trigger = GPIOPinConfig.TRIGGER_RISING_EDGE initValue = ignored

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
6	BUTTON 2	Tamper/Key (B3)	controllerNumber = 2 pinNumber = 13 direction = GPIOPinConfig.DIR_INPUT_ONLY mode = GPIOPinConfig.MODE_INPUT_PULL_UP trigger = GPIOPinConfig.TRIGGER_FALLING_EDGE initValue = ignored
50	IO_EXPANDER_INT	Interrupt pin for on-board STMPE1600	controllerNumber = 8 pinNumber = 8 direction = GPIOPinConfig.DIR_INPUT_ONLY mode = GPIOPinConfig.MODE_INPUT_PULL_UP trigger = GPIOPinConfig.TRIGGER_FALLING_EDGE

WARNING: The information in this appendix is presented in addition to the board schematics provided by the manufacturer. Be sure to familiarize yourself with the manufacturer's documentation for all of the GPIO pins. Failure to do so can result in damaged peripherals or even a damaged embedded board.

Additional notes:

- If the `controllerNumber` is set as `GPIOPinConfig.DEFAULT`, it is interpreted as 6. The `pinNumber` in this case is 6 as well, this a `GPIOPin` connected to the Green LED will be opened by default.
- If the `mode` is set as `GPIOPinConfig.DEFAULT` and `direction` is set as `INPUT` then the default `mode` is interpreted as `MODE_INPUT_PULL_DOWN`. If the `direction` is set as `OUTPUT`, then the default mode is interpreted as `MODE_OUTPUT_OPEN_DRAIN`.
- The following GPIO pins **cannot** be opened because they are mapped to other resources. For more information, refer to evaluation board schematics from ST Micro.
  - SPI bus: PF9, PF8, PF7, PF6
  - ADC: PF10
  - UART: PA9, PA10
  - SDHC: PC8, PC9, PC10, PC11, PD2
  - I2C: PB6, PB9
  - Ethenet: PC3, PG13, PG14, PC2, PB8, PG11, PA1, PC4, PC5, PH6, PH7, PA7, PB10, PA3
  - JTAG/SWO: PA13, PA14

  - USB (OTG-HS): PB0, PB1, PB5, PB10 ~ PB13, PA3, PI11
  - LCD: PJ0 ~ PJ15, PK0~PK15, PI12~PI15
  - FMC: PF0 ~ PF5, PF11 ~ PF15, PG0 ~PG5, PH8~PH15, PI0~PI3, PI9, PE0, PE1
  - SDRAM: PH5, PG15, PF11, PH3, PH2, PG8
- Onboard LEDs are connected with their anodes to 3v3. To turn them on, set the pin to false.

GPIO Ports

The following GPIO ports are pre-configured:

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
8	LEDS	Green LED, Orange LED, Red LED, Blue RED	direction = GPIOPinConfig.DIR_OUTPUT_ONLY initValue = 0 For each pin information are same as the GPIO Pins section above.

Additional notes:

- The following GPIO ports are supported: PA, PB, PC, PD, PE, PF, PG, PH, PI, PK

Inter-Integrated Circuit (I2C)

The following I2C devices are pre-configured:

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
500	IO_EXPANDER	On-board IO Expander, STMPE1600QTR. (Using this IO Expander, the on-board Joystick can be accessed indirectly)	controllerNumber = 1 address = 66 addressSize = 7 clockFrequency = 100000

The following additional hardware information is applicable for I2C devices:

- The CMSIS I2C driver on this board only supports the following two types of clock frequencies: 100kHz (`ARM_I2C_BUS_SPEED_STANDARD`), and 400kHz (`ARM_I2C_BUS_SPEED_FAST`).
- `I2C1` controller information: `I2C1_SCL` maps to PB6, and `I2C1_SDA` maps to PB9. The address capability is 7-bit only, the addresses used are 0x32 (on-board Audio) and 0x42 (on-board IO Expander), and the clock frequency is either 100 KHz (standard speed) or 400 KHz (fast speed).

The following additional information is applicable for I2C devices on the board itself:

- For the IO Expander: the device is `STMPE1600`, which is a GPIO port expander that is able to interface a main digital ASIC(MCU) via the two-line bidirectional bus (I2C). The connected MCU's I2C Controller is: I2C1; applicable pins are `I2C1_SCL` :

PB6, I2C1_SDA : PB9, and IO_EXPANDER_INT : PI8 (the IO Expander interrupt pin is connected to the MCU). The clock frequency may be between: 0 and 400 kHz.

- Joystick: Remember that the pins are not connected to MCU, but are instead connected to the IO Expander. By reading and writing the registers of the IO Expander, the joystick status can be monitored. Out of 16 pins of IO Expander, the following 5 pins are connected to the joystick: EXP_IO14: SELECT; EXP_IO13: DOWN; EXP_IO12: LEFT; EXP_IO11: RIGHT; EXP_IO10: UP.

SPI

SPI is currently supported, however there are no pre-configured SPI devices on the board. However, the board is able to connect an external module to the pinout on the board as a ad-hoc device, as follows.

- SPI5 is the only connection supported, so the controllerNumber should be 5. This is default value. Use the following SPI pins: MOSI: PF9; MOSI: PF8; CLK: PF7; CS: PF6.
- CS is available only as 'active low'
- According to the ST Micro documentation, SPI5 can communicate at up to 45 Mbits/s.
- The following are default parameters:
  - controllerNumber : 5
  - address: 0
  - clockFrequency : 10000000 (10 MHz)
  - wordLength : 8 bits
  - bitOrdering : Big Endian
  - csActive : ACTIVE_LOW
  - clockMode : CPOL-High , CPHA-Even

UART

The following UART devices are pre-configured:

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
5000	USART1	Onboard USART1 connector (CN8)	controllerName = USART1 baudRate = 115200 parity = 0 dataBits = 8 stopBits = 1 flowControl = 0

Note the following.

- Supported UART: USART1 at onboard hardware connector CN8.
- Hardware setting: jumper JP7 should be set to 1-2 (default setting)

- Supported `parity`: `none`, `even` and `odd`. `mark` and `space` are not supported
- Supported `databit`: 5, 6, 7, 8 and 9
- Supported `stopbit`: 1, 1_5 and 2
- Hardware flow control is not supported
- Log channel note: UART logging is disabled by default. If the internal property `log.method` is set to `UART`, then USART1 is dedicated to logging and you cannot use USART1 as the Device I/O UART
- The following are default parameters:
  - `baudrate`: 115200
  - `parity`: none
  - `databit`: 8
  - `stopbit`: 1
  - flow control: none

Watchdog

The following watchdog devices are pre-configured:

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

- Note that the platform watchdog is not a DIO watchdog for Java application. Instead, it's the platform's general watchdog.
- The `system.watchdog` property must be enabled. This is an internal property for platform watchdog in the `jwc_prop.ini` file. Note that the default value is `false`.
- The timeout value is 4 seconds, which means the device will reboot if it is frozen for more than 4 seconds.

Glossary

Access Point

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

ADC

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

AMS

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

APDU

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

API

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

ARM

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

AT commands

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

AXF

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

BIP

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

CDMA

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

CLDC

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

Configuration

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

DAC

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

ETSI

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

GCF

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

GPIO

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

GPIO Port

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

GSM

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

HTTP

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

HTTPS

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

ICCID

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

IMP-NG

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

IMEI

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

IMlet

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

IMlet Suite

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

IMSI

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

I2C

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

ISA

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

JAD file

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

JAR file

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

JCP

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

JDTS

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

Java ME platform

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

JSR

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

Java Virtual Machine

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

KVM

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

LCDUI

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

MIDlet

An application written for MIDP.

MIDlet suite

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

MIDP

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

MSISDN

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

MVM

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

Obfuscation

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

Optional Package

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

Preemption

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

Preverification

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

Profile

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

Provisioning

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

Pulse Counter

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

Push Registry

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

RISC

Reduced Instruction Set Computing. A CPU design based on simplified instruction sets that provide higher performance and faster execution of individual instructions. The ARM architecture is based on RISC design principles.

RL-ARM

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

RMI

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

RMS

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

RTOS

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

RTSP

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

SCWS

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

SD card

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

SIM

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

Slave Mode

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

Smart Card

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

SMS

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

SMSC

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

SOAP

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

SPI

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

SSL

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

SVM

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

Task

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

TCP/IP

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

Terminal Profile

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

UART

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

UICC

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

UMTS

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

URI

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

USAT

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

USB

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

USIM

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

WAE

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

WAP

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

Watchdog Timer

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

WCDMA

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

WMA

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

XML Schema

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