Oracle® Java ME Embedded

Getting Started Guide for the Reference Platform (Keil)

Release 3.3.1

E38140-02

June 2013

This book describes how to install and run the Oracle Java ME Embedded software on the Keil MCBSTM32F200 reference platform, using the RTX operating system.

ORACLE®

Oracle Java ME Embedded Getting Started Guide for the Reference Platform (Keil), Release 3.3.1

E38140-02

Contents

List of Examples

List of Figures

List of Tables

Preface

This book describes how to install Oracle Java ME Embedded software onto a Keil MCBSTM32F200 embedded device. Readers using this guide must be familiar with the *Information Module Profile - Next Generation (IMP-NG) 1.0 Specification*.

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	`$`
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

Running the Oracle Java ME Embedded Software on the Keil MCBSTM32F200 Board

The Keil MCBSTM32F200 board is the reference device of the Oracle Java ME Embedded software. This chapter documents how to install the Java ME Embedded software with RTX binary onto the Keil MCBSTM32F200 board, configuring the Java ME Embedded system, connecting to the command-line and logging interfaces, and installing and running a Java ME Embedded "IMlet" application.

The following items are required for developing on the Keil MCBSTM32F200 board:

- The Oracle Java ME Embedded RTX Software Distribution, Version 3.3.1
- A desktop computer running Windows XP or later with two USB ports
- A Keil MCBSTM32F200 Cortex Board
- A hardware debugger, such as the ULINK-ME or ULINK 2
- A MicroSD card (with an SD adapter, if necessary for connecting to the desktop computer)
- A networking LAN cable with RJ-45 interface
- 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 4.54, 4.6, or 4.7 of the MDK-ARM development kit. The MDK-ARM development kit can be obtained from the following site:

`https://www.keil.com/download/product`

Be sure the download completes successfully: for example, the file size for the MDK-ARM Lite version is approximately 500 MB.

Once downloaded, install the MDK-ARM tool by double-clicking on the executable. After the tool install finishes, verify the installation by double-clicking on the Keil µVision 4 IDE executable.

WARNING: There is currently a bug in Version 4.60 of the MDK-ARM Development Kit that prevents an output file from being generated during binary flashing onto the Keil boards. If this occurs, please use a different version of the MDK-ARM Development Kit.

Downloading and Installing the PuTTY Terminal Emulator Program

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

`http://www.putty.org/`

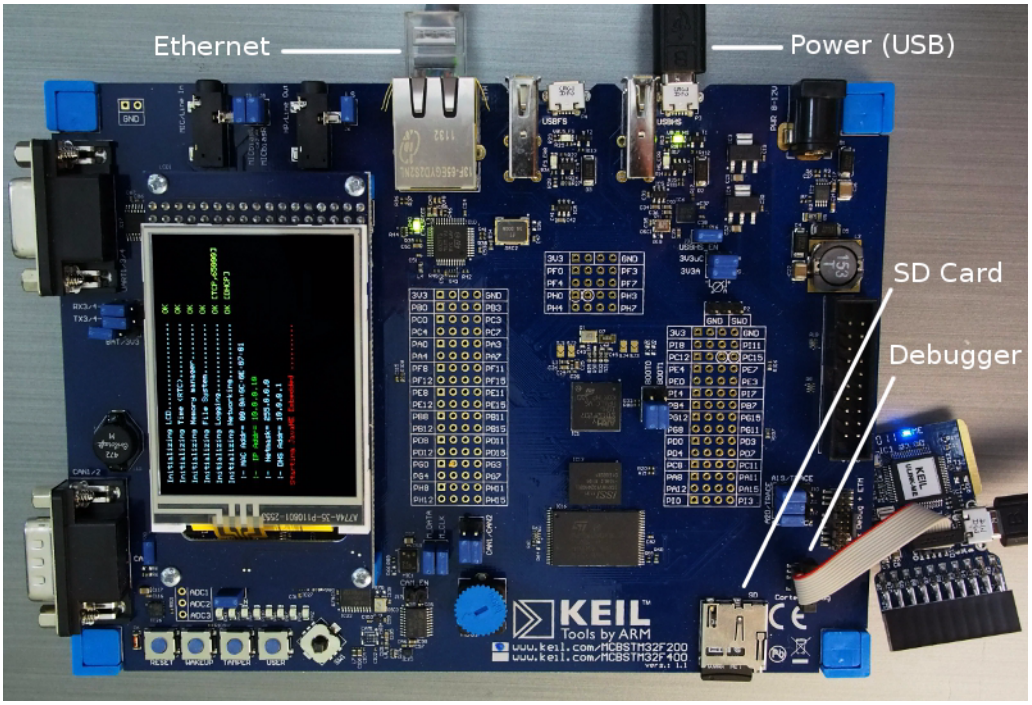
The terminal emulator executable is directly downloadable as `putty.exe`. The terminal emulator is used to connect to two separate sockets: one for the command-line interface (CLI) that issues commands to the board, and one for the logging or system output provided by the board.

Connecting the Keil MCBSTM32F200 Board to the Computer

The Keil MCBSTM32F200 board comes with two USB cables. Both cables must be connected to the desktop computer. The first USB cable attaches to the USBHS connector, and provides power to the board, as shown in Figure 1–1. The other connects to the hardware debugging unit, such as the ULINK-ME, which in turn connects to a compatible debugging port on the evaluation board. See the instructions that come with your specific hardware debugging unit for more information.

After the USB cables are connected, ensure that the board and the debugger are receiving power by verifying that the LEDs on both the board near the USB connection and the hardware debugger are lit. The LED near the USB connection should be lit green. With the Keil MCBSTM32F200, the backlight of the touchscreen display also lights up, even though it might not be displaying any information. See Figure 1–1.

Figure 1–1 The Keil MCBSTM32F200 Board



The first time the board is connected to the desktop computer, Windows typically installs the device driver software for the debugger. The ULINK-ME, for example, uses the standard set of USB drivers that comes with Microsoft Windows.

Setting Up the MicroSD Card

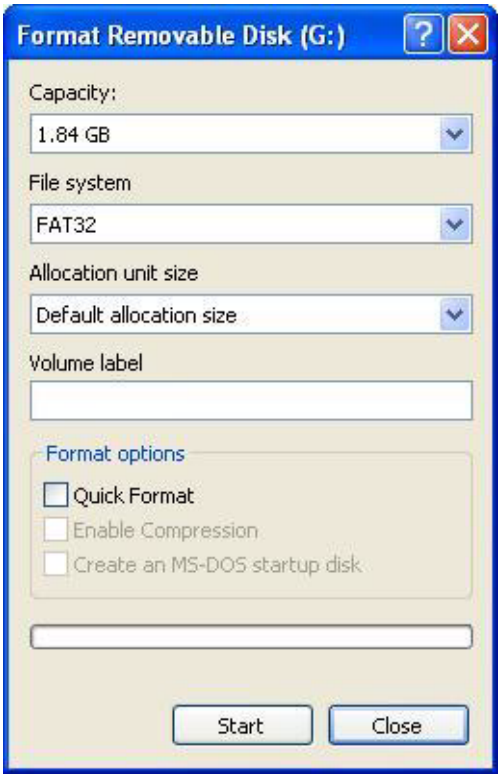
The MicroSD card must contain configuration files used by the Java ME Embedded native distribution, including the initialization properties and security policy files. Ensure that you are using a compatible SD card. This release has been tested with the Transcend 2 GB card. For a list of supported cards, please refer to the appropriate documentation for the RTX OS and the Keil F200 series embedded boards.

Note: All filenames on the MicroSD card *must* adhere to the 8.3 file format.

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 1–2. The Volume label is optional. Press the **Start** button.

Figure 1–2 Windows Formatter Settings



3. Once the formatting completes, copy all the files inside the directory `sd_card/` in the Oracle Java ME Embedded distribution to the root directory of the SD card.
4. Edit the file `rtc_upd.cfg` on the root directory of the SD card. (This can be done by opening the file with a standard text editor.) Edit the current time and date that the device should use, in the format `YYYY/MM/DD HH:MM:SS Zone`. For example, the file `rtc_upd.cfg` could contain the following:

```
2012/11/22 18:46:00 GMT
```

Next, rename the file `rtc_upd.cfg` to `rtc.cfg`.

5. Open the file `platform.cfg` on the root directory of the SD card to specify the appropriate networking information for the device.

   Edit the MAC address if you have more than one board in your local network or if you think that there is a MAC address conflict. The following is an example of a MAC address:

```
#-------------------------------------------
# MAC Address
# - It should be unique value in the same LAN
#-------------------------------------------
mac.addr= 00:0A:0C:0E:07:0A
```

   Note: The MAC address must comply with the IEEE 802.3 standard.

   If your network uses dynamic IP addresses, uncomment the DHCP line under Dynamic IP Address and comment the lines under Static IP Address. If your network uses static IP addresses, uncomment and edit the lines under Static IP Address and comment the line under Dynamic IP Address.

   The following is an example of configuring options for using static IP addresses:

```
#-------------------------------------------
# Dynamic IP addess (DHCP)
#-------------------------------------------
#ip.method= dhcp

#-------------------------------------------
# static IP addess
#-------------------------------------------
ip.method= static
ip.addr= 10.0.0.10
ip.netmask= 255.0.0.0
ip.gateway= 10.0.0.1
ip.dns= 10.0.0.1
```

6. In the file `platform.cfg`, disable the watchdog by setting its value to `false`:

```
#-------------------------------------------
# H/W Watchdog
#-------------------------------------------
watchdog.enable= false
```

   The watchdog resets the system if a program appears to be stuck. However, the watchdog may trigger while the board communicates with the Oracle ME SDK and NetBeans, especially during debugging.

7. If you plan to use on-device debugging (ODD) with an IDE, edit the file `jwc_prop.ini` in the `java/` directory of the SD card. Set the property `odt_run_on_start` to `true` to enable ODD.

Installing the Firmware on the Evaluation Board

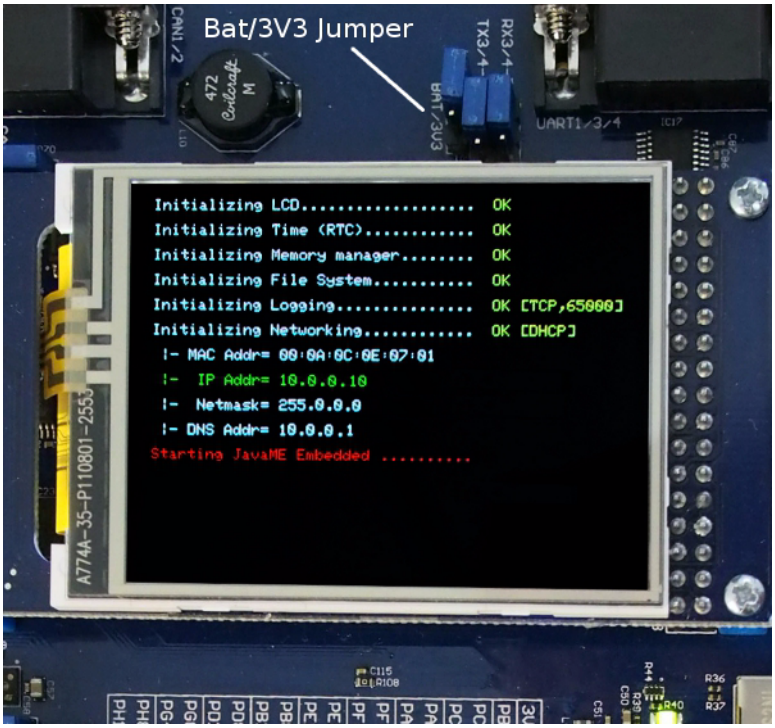
The Java ME Embedded distribution contains a utility that uses the MDK-ARM software to erase and flash the Oracle Java ME Embedded binary onto the evaluation board. Use the following procedure to prepare the SD card and install the firmware:

1. Use the **Safely Remove Hardware** function on the Windows desktop to remove the SD card from the computer. (In Windows XP, for example, the Safely Remove Hardware 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. Be sure that the board is not powered up by temporarily disconnecting the USB cables. Insert the MicroSD card into the SD card slot on the board.
3. Switch the "BAT/3V3" jumper near the RS-232C connection on the board to the left (BAT) position, outlined in Figure 1–3.
4. Connect the board to the Windows desktop computer as you did previously, including the network cable, the USB power cable, and the debugger with its USB cable. The board should now be receiving power.
5. On the desktop, check the file `flash.bat` inside the directory `flash/` in the Oracle Java ME Embedded software. Ensure that this file contains the correct path to the MDK-ARM tool binary. For example:

   ```
   C:\Keil\UV4\UV4.exe -f jmee.uvproj -o output.txt
   ```

6. With the board connected, run the batch file `flash.bat` to download the distribution to the evaluation board. The MDK-ARM tool notifies you when the download has completed. If the MDK-ARM tool requires a Flash Upgrade of the debugger, press **OK** to allow this, then restart the `flash.bat` when completed.
7. 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 1–3. Note that after the time clock has been initialized, the runtime will rename the `rtc.cfg` file on the MicroSD card back to `rtc_upd.cfg` and ask you to reset the board to activate the clock's settings.

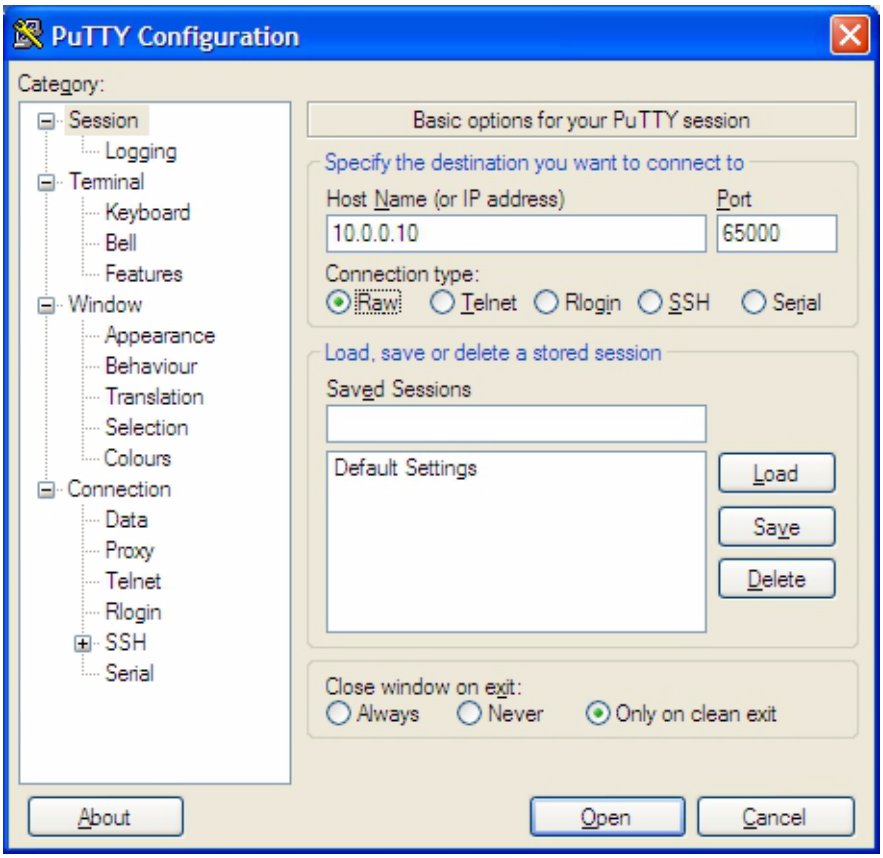
Figure 1–3 The MCBSTM32F200 Touchscreen After Installing Java ME Embedded



Connecting to the CLI and Logging Ports

When the Java ME Embedded distribution is running, note the IP address that is shown on the touchscreen in green, and start two PuTTY executables. Use these to create raw socket connections to the IP address and two ports shown in Table 1–1. For example, the PuTTY settings to initiate a connection to the IP address of 10.0.0.10 and the port 65000 is shown in Figure 1–4. Note that the IP address of your embedded board may be different.

Figure 1–4 PuTTY Options for Connecting to the Keil Board



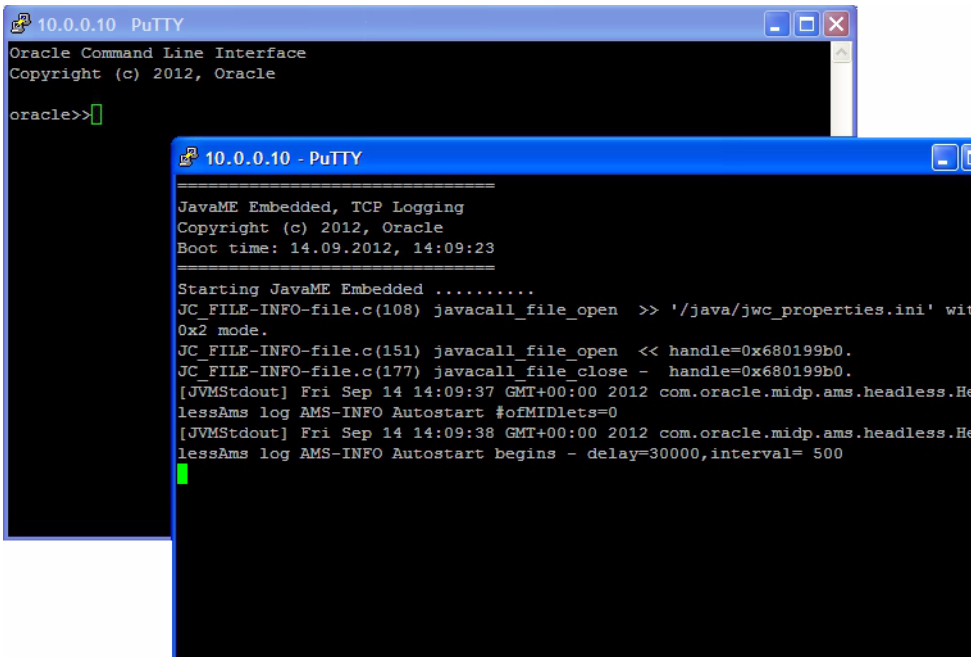
The two default ports that are used are shown in Table 1–1.

Table 1–1 Ports Used by the Embedded Board

Port	Description
65000	Logging / Java VM System Output
65002	Command-line interface

The window that outputs data from port 65000 provides logging and standard output information from the device. This is useful when determining if a Java ME embedded program ran successfully on the board. The window from port 65002 provides a command-line interface (CLI). Both the CLI and logging windows are shown in Figure 1–5. The logging port is defined in the jwc_prop.ini properties file using value assigned to the `log.tcp.port` property.

Figure 1–5 Logging and Command Line Interfaces



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

In the CLI window, you can run the Application Management System (AMS) commands shown in Table 1–2. Note that the AMS syntax may change in future releases; entering `help [command]` is the best way to obtain the latest CLI syntax.

Table 1–2 AMS CLI Commands

Syntax	Description
`ams-list [INDEX or NAME|VENDOR]`	List all installed IMlet suites and their statuses or show the detail of a single suite
`ams-install <URL> [username:password]`	Install an IMlet using the specified JAR or JAD file, specified as a URL. An optional username and password can be supplied for login information as well.
`ams-update <INDEX or NAME|VENDOR>`	Update the installed IMlet
`ams-remove <INDEX or NAME|VENDOR>`	Remove an installed IMlet
`ams-run <INDEX or NAME|VENDOR> [IMLET_ID] [-debug]`	Execute the specified IMlet or the default if none is specified. An optional debug parameter can be specified 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

Table 1–2 (Cont.) AMS CLI Commands

Syntax	Description
`ams-suspend <INDEX or NAME|VENDOR> [IMLET_ID]`	Suspend (pause) the specified IMlet or the default if none is specified
`ams-resume <INDEX or NAME|VENDOR> [IMLET_ID]`	Resume the specified IMlet or the default if none is specified
`ams-setup <INDEX or NAME|VENDOR>`	Display the setup menu of the IMlet
`ams-info <INDEX or NAME|VENDOR>`	Show information about the installed IMlet
`ams-log <command> [param1, param2, ..., paramN]` `ams-log wdog`	Display the IMlet log or watchdog log if recorded by the watchdog handler in the platform
`ams-logger-list [INDEX or NAME|VENDOR]`	Retrieve the logger list for the IMlet or all the tasks if one is not specified
`ams-logger-info <INDEX or NAME|VENDOR> [LOGGER_NAME]`	Retrieve logger info for the specified IMlet and logger or all the loggers if is one is not specified
`ams-logger-level-set <INDEX or NAME|VENDOR> [LOGGER_NAME] <LOGGER_LEVEL>`	Set the logger level for specified IMlet or all loggers if one is not specified
`help [command name]`	List the available commands or detailed usages for a single command
`sysmenu <on PASSWORD|off>`	Enable hidden system menu commands. Currently, the password is `12345`.
`exit`	Terminates the current session.

When the `sysmenu` command is entered with the on option, additional system menu commands are available with the AMS CLI, as shown in Table 1–3.

Table 1–3 Additional System Commands Available in the AMS CLI

Syntax	Description
`setprop <KEY> <VALUE>`	Sets a property identified by `<KEY>` with the value `<VALUE>`
`getprop <KEY>`	Returns a property identified by `<KEY>`
`odd [on|off]`	Explicitly sets the on-device debugging (ODD) property to on or off. If no parameters are passed, returns the current ODD value.
`shutdown [-r]`	Perform either a shutdown of the board, or a reboot if the `-r` parameter has been passed. Note that the watchdog should be enabled in the `platform.cfg` file to successfull reboot. (See Setting Up the MicroSD Card)

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

Note: Note that the AMS syntax and output may change in future Oracle Java ME Embedded releases.

First, install the application using the `ams-install` command, specifying its location either on the MicroSD card (using the `Memorycard` directory, which maps to `memcard` on the filesystem) or across a network using a URL.

```
oracle>> ams-install file:///Memorycard/hello.jar
<<ams-install,start install,file:///Memorycard/hello.jar
<<ams-install,install status: stage 0, 5%
<<ams-install,install status: stage 3, 100%
<<ams-install,install status: stage 4, 100%
<<ams-install,OK,Install success

oracle>> ams-install http://www.example.com/netdemo.jar
<<ams-install,start install,http://www.example.com/netdemo.jar
<<ams-install,install status: stage 0, 5%
<<ams-install,install status: stage 3, 100%
<<ams-install,install status: stage 4, 100%
<<ams-install,OK,Install success

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

Note that the final installation example failed with an error code and matching description. If the install process shows any error code, see Table B–1 in Appendix B for more information on how to resolve the error.

Once an IMlet is installed, verify it using the `ams-list` command. Here, we have added an additional IMlet: `rs232dem`. Each IMlet has been assigned a number for convenience.

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

The `ams-remove` command can be used to remove any installed IMlet.

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

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

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

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

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

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

Subsequent chapters will discuss how to use either the NetBeans or Eclipse IDE to develop, install, and test applications directly on the board.

Additional Methods of Obtaining Logging Information

There are several additional ways to connect to the embedded board and obtain logging output. These include the following:

1. **UART - Through RS232**

   To utilize this method, connect to the board using an RS232 serial cable and use a tool such as PuTTY to create a serial connection from a terminal on the Windows desktop using the following options:

   Baud rate: 115200

   Data bits: 8

   Stop bits: 1

   Parity: None

   Flow control: XON/XOFF

   Also, ensure that the `log.method` property from the `jwc_prop.ini` file contains the value `UART` in its list.

2. **USB/COM - Through a USB COM port and an appropriate driver**

   In order to use this method, connect a USB cable from your desktop to the Keil board. A USB to COM driver must also be installed on your Windows desktop machine. Drivers' installers and installation instructions can be found in the driver directory of the Keil bundle. Finally, ensure that the `log.method` property from the `jwc_prop.ini` file contains the value `USBCOM` in its list. Then, create a serial connection from your terminal with the following options:

   Baud rate: 115200

   Data bits: 8

   Stop bits: 1

   Parity: None

   Flow control: XON/XOFF

3. **ITM - Through ITM trace supported by MCU**

   An instrumentation trace can be used as supported by the ARM microcontroller. See the embedded board documentation for more information on this method. Also, ensure that the `log.method` property from the `jwc_prop.ini` file contains the value `ITM` in its list.

2

Using NetBeans or Eclipse with the Keil MCBSTM32F200 Board

Developers can run and debug IMlets on the Keil MCBSTM32F200 board directly from the NetBeans IDE or Eclipse IDE using the Oracle Java ME SDK. This chapter describes how to add the board to the Device Selector in the Oracle Java ME SDK and how to debug an IMlet on the board from both the NetBeans IDE and the Eclipse IDE.

Using NetBeans with the Keil MCBSTM32F200 Board

Running and debugging IMlet projects on the Keil MCBSTM32F200 board using the NetBeans IDE requires the following software:

- NetBeans IDE 7.3 with Java ME enabled
- Oracle Java ME SDK
- Oracle Java ME SDK NetBeans Plugin

Install the Oracle Java ME SDK Plugin for NetBeans

After installing NetBeans, use these steps to install the remaining software.

1. Ensure that Java ME is enabled in NetBeans. This can be done by selection **Tools** -> **Plugins** and selecting the **Installed** pane. Activate the Java ME plugin if it is not already activated.
2. Install the Java ME SDK distribution. See the Java ME SDK documentation for details.
3. Install the Oracle Java ME SDK NetBeans plugin. This is a downloadable ZIP file that consists of a number of NetBeans modules (`.nbm` files) that can be added using the **Tools** -> **Plugins** dialog and selecting the **Downloaded** pane. The Oracle Java ME SDK NetBeans plugin is required to use the Device Selector to connect to the board. See the Oracle Java ME SDK Release Notes for installation instructions:

   `http://docs.oracle.com/javame/dev-tools/jme-sdk-3.3/release-notes/toc.htm`
4. Ensure that the Oracle Java ME Embedded 3.3 appears in the list of Java ME platforms. In the NetBeans IDE, go to **Tools** -> **Java Platforms**. If the Oracle Java ME Embedded 3.3 does not appear in the list of J2ME platforms, follow these steps:
   - Click on **Add Platform**.
   - Select **Java ME CLDC Platform Emulator** and click Next.

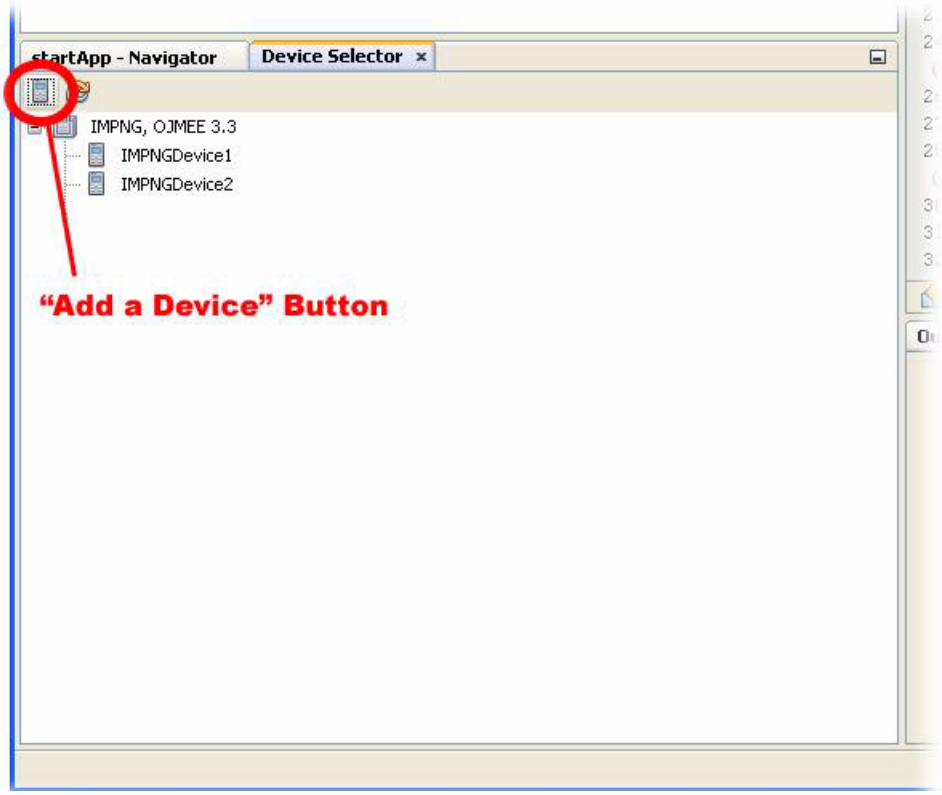
- Select the folder where the Oracle Java ME SDK 3.3 runtime for Keil MCBSTM32F200 resides and follow the instructions to install it. Then, click **Finish** to close the dialog.

5. Ensure that the Keil MCBSTM32F200 board is running the Oracle Java ME Embedded distribution. See Chapter 1 for more information on how to install the runtime distribution on the Keil MCBSTM32F200 board.

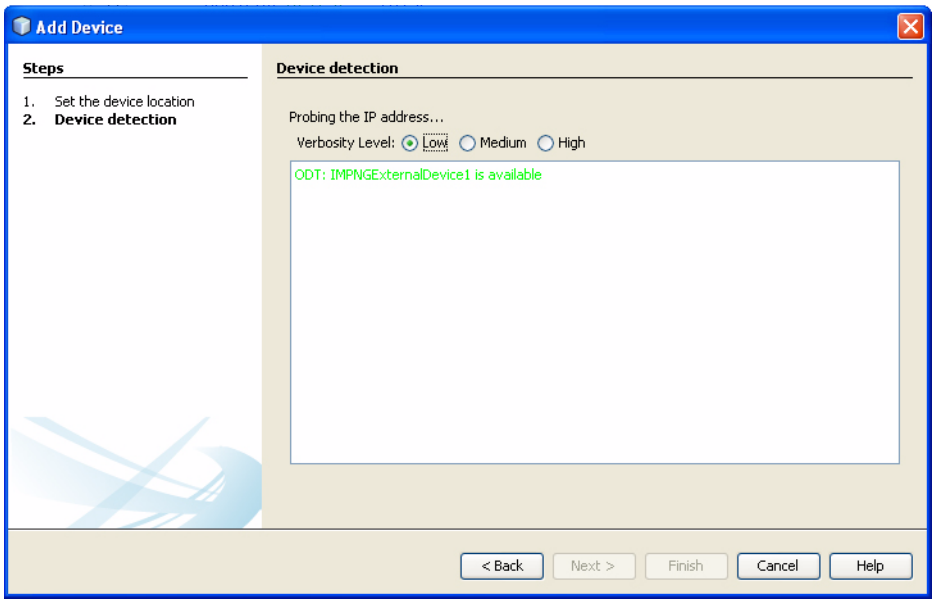
Adding the Keil MCBSTM32F200 Board to the Device Selector

Follow these steps to add the board to the Device Selector in the Oracle Java ME SDK:

1. Ensure that the property `odt_run_on_start` is set to true in the file `java/jwc_prop.ini` on the Keil MCBSTM32F200.
2. Ensure that the board is receiving power and the Oracle Java ME Embedded software is running.
3. If necessary, open TCP port 55123 in the firewall settings of your computer. The exact procedure to open a port differs depending on your version of Windows or the firewall software that is installed on your computer.
4. Start the NetBeans IDE. In the NetBeans IDE, go to **Tools** -> **Java ME** -> **Device Selector**
5. On the Device Selector, click on the **Add a Device** button at the top of the Device Selector window.



6. Write the IP address of the Keil MCBSTM32F200 board in the **IP Address** field and click **Next**. You can find the IP address of the Keil MCBSTM32F200 board by looking at the touchscreen on the board.



7. Once the device is detected, click **Finish** on the Device Detection screen.

The list of devices in the Device Selector should now include `IMPNGExternalDevice1`.

Assigning the Keil MCBSTM32F200 Board to Your Project

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

1. Right-click on your project and choose **Properties**.
2. Select the **Platform** category on the properties window.
3. Select **IMPNGExternalDevice1** from the device list.

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

1. Select **File** -> **New Project**.
2. Select the **Java ME** category and **Embedded Application** in Projects. Click **Next**.
3. Provide a project name and click **Next**. Be sure that the **Create Default Package and IMlet Class** option is checked.
4. Ensure the Emulator Platform is **Oracle Java ME Embedded 3.3**. Then, select **IMPNGExternalDevice1** from the device list and click **Finish**.

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

Sample Source Code

Once the project is created, use the source code in Example 2–1 for the default `IMlet.java` source file.

Example 2–1 Sample Code to Access a GPIO Port

```
package embeddedapplication1;

import com.oracle.deviceaccess.PeripheralManager;
```

```
import com.oracle.deviceaccess.PeripheralNotAvailableException;
import com.oracle.deviceaccess.PeripheralNotFoundException;
import com.oracle.deviceaccess.gpio.GPIOPin;
import java.io.IOException;
import javax.microedition.midlet.*;


public class IMlet extends MIDlet {

    public void startApp() {

        try {
            GPIOPin pin = (GPIOPin)PeripheralManager.open(2);
            boolean b = pin.getValue();
        } catch (PeripheralNotAvailableException ex) {
            ex.printStackTrace();
        } catch (PeripheralNotFoundException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }

    }

    public void pauseApp() {
    }

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

This sample application will obtain an object representing GPIO pin 1 from the `PeripheralManager`, and attempt to obtain its high/low value.

Accessing the Peripherals on the Keil MCBSTM32F200

There are two ways to allow access to the peripherals on the Keil MCBSTM32F200. The first is to use unsigned applications and modify the security policy file, and the second is to digitally sign the application with the appropriate API permissions requested in the JAD file.

Method #1: Modifying the Security Policy File

Modifying the security policy file is only necessary in the event that a user must manually install the application on the board, at which point the unsigned application will be installed in the `untrusted` security domain.

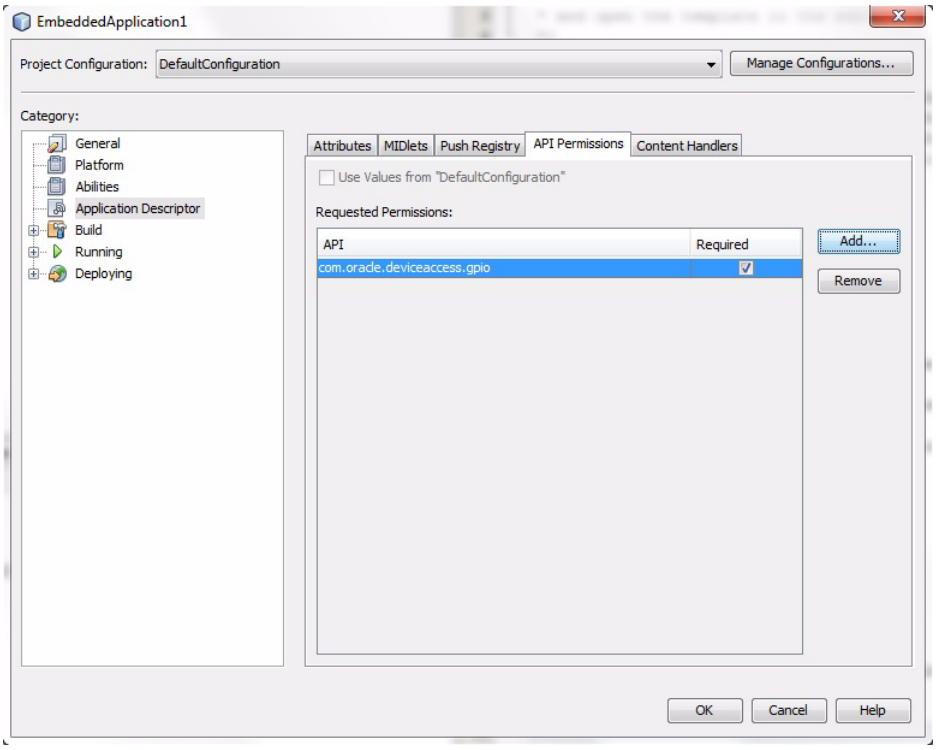
With this method, simply add the line "`allow: device_access`" to the "`untrusted`" domains of the security policy file. By default, this is located on the SD card in the `appdb/_policy.txt` file, but be sure to check the `security.policy` file entry in the `java/jwc_prop.ini` file to verify the current file name.

Note that if an application is installed on the board using NetBeans or Eclipse during development, the application will automatically be installed in the `maximum` security domain as a convenience. Manual installation, however, will install the unsigned application into the `untrusted` security domain. Note that after development is finished, you should publish your applications with signed API permissions.

Method #2: Signing the Application with API Permissions

The second method is more complex, but is the preferred route for production applications that are widely distributed. First, the JAD file must have the proper API permissions. Right-click the project name (**EmbeddedApplication1** in this example) and choose **Properties**. Select **Application Descriptor**, then in the resulting pane, select **API Permissions**. Click the **Add...** button, and add the `com.oracle.deviceaccess.gpio` API, as shown in Figure 2–1. Click **OK** to close the project properties dialog.

Figure 2–1 Adding API Permissions with NetBeans



Applications that access the Device Access APIs must also be signed. Here are the instructions on how to setup a keystore with a local certificate that can be used to sign the applications.

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

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

   This command will generate 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.

2. Modify the `appdb/_main.ks` keystore file from the Keil Micro SD card by performing the following `mekeytool.exe` command (or alternatively `java -jar MEKeyTool.jar...` if your distribution contains only that) that ships with the Oracle Java ME SDK 3.3 distribution.

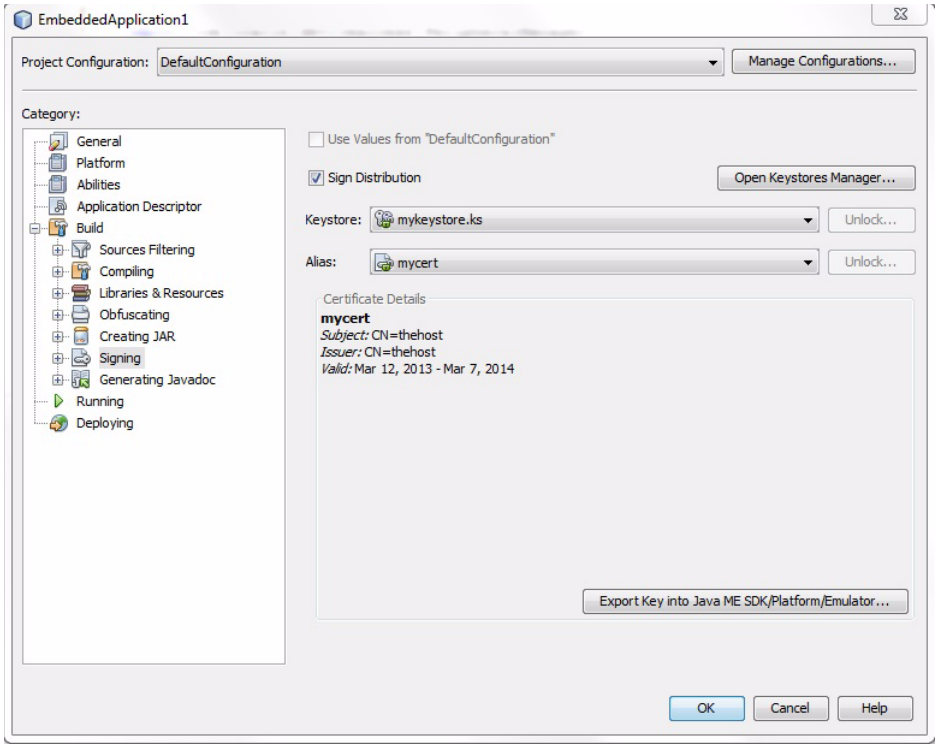
   ```
   {mekeytool} -import -MEkeystore _main.ks -keystore mykeystore.ks
   -storepass spass -alias mycert -domain trusted
   ```

This will import the information in `mykeystore.ks` you just created to the _ `main.ks` keystore.

Use the following steps to sign your application before deploying to the Keil MCBSTM32F200 board.

1. Right click your project and select **Properties**.
2. Choose the **Signing** option under the **Build** category.
3. Open the **Keystores Manager** and import the `mykeystore.ks` file that you created.
4. Check the **Sign Distribution** box. If you wish, unlock the keystore and the key with the passwords that you specified earlier. This is shown in Figure 2–2.
5. When the project is built and run, it will be digitally signed and deployed to the Keil MCBSTM32F200.

Figure 2–2 The Signing Pane in the NetBeans Project Properties



Debugging an IMlet on the Keil MCBSTM32F200 Board

Follow these steps to debug an IMlet using NetBeans:

1. Open your IMlet class on the NetBeans editor.
2. Click once directly on the line number where you want to set a breakpoint. The line number is replaced by a red square and the line is highlighted in red.
3. Select **Debug** -> **Debug Project** or use the Debug button on the toolbar.

The debugger connects to the debug agent on the board and the program execution stops at your breakpoint, as shown in Figure 2–3.

Figure 2–3 Debugging an IMlet on the Board Using NetBeans

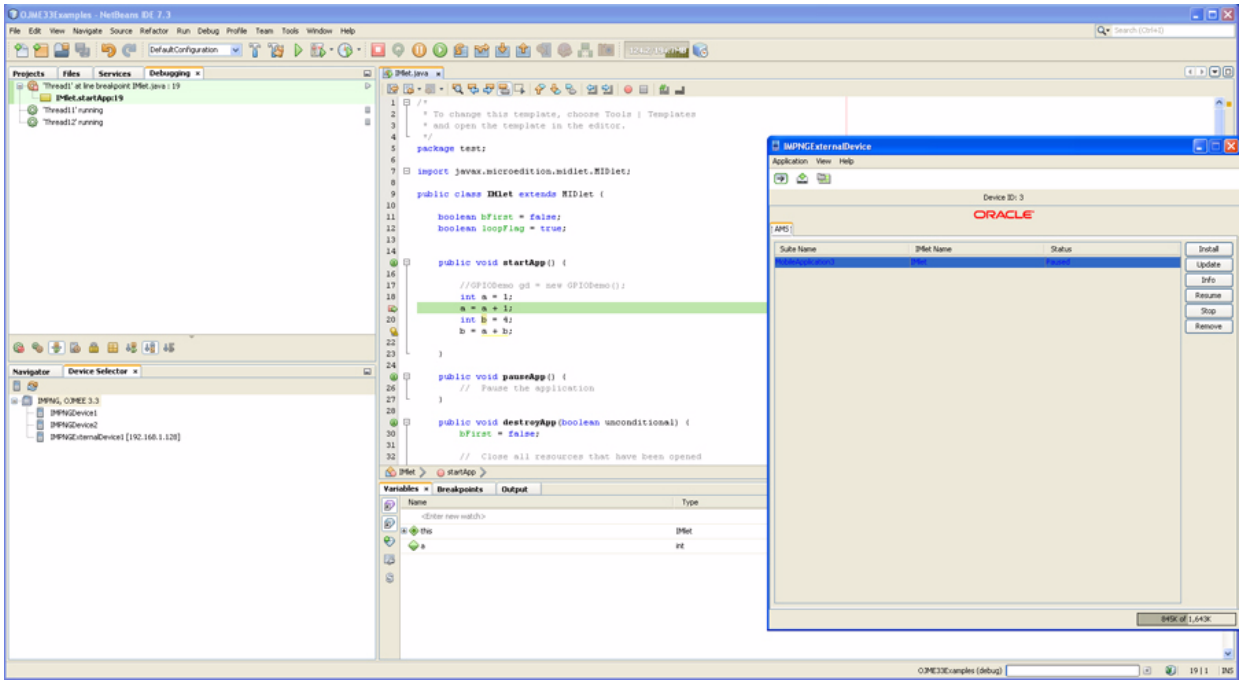


Figure 2–3 shown an entire NetBeans debugging environment that allows the programmer to execute a program step by step as well as add and remove variables from a watch list on the bottom of the screen.

For more information on using the device access APIs, please see the Device Access API Guide and the associated javadocs.

Using Eclipse with the Keil MCBSTM32F200 Board

Running and debugging IMlet projects on the Keil MCBSTM32F200 board using the Eclipse IDE requires the following software:

- Eclipse 3.7 Indigo or Eclipse 4.2 Juno
- Oracle Java ME SDK
- Oracle Java ME SDK Eclipse Plugin

Install the Oracle Java ME SDK Plugin for Eclipse

After installing Eclipse, use these steps to install the remaining software.

1. Install the Java ME SDK distribution. See the Java ME SDK documentation for details.
2. Install the Oracle Java ME SDK Eclipse plugin. This is required to use the Device Selector to connect to the board. See the Oracle Java ME SDK Release Notes for installation instructions:

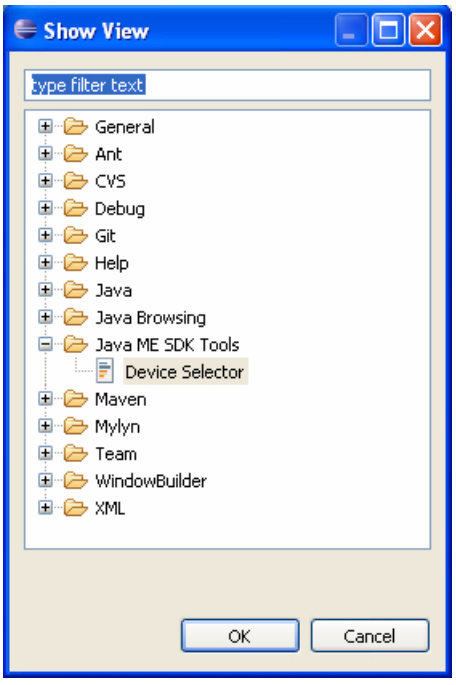
   `http://docs.oracle.com/javame/dev-tools/jme-sdk-3.3/release-notes/toc.htm`
3. Ensure that the Keil MCBSTM32F200 board has the Oracle Java ME Embedded 3.3 runtime. See Chapter 1 for more information on how to install the runtime distribution on the Keil MCBSTM32F200 board.

4. Ensure that the Oracle Java ME Embedded 3.3 appears in the list of Java ME platforms. If it doesn't appear, follow these steps for your project properties.
   - Under the **Java ME** category, select **Device Management**. In the Device Management window, press the **Manual Install...** button.
   - The Manual Device Installation window appears, without the Oracle Java ME Embedded devices. Press the **Browse** button. A browser window appears.
   - Browse to the base directory of the Java ME SDK environment and press the **OK** button. After the platform is scanned and the devices are installed, close each of the respective dialogs.

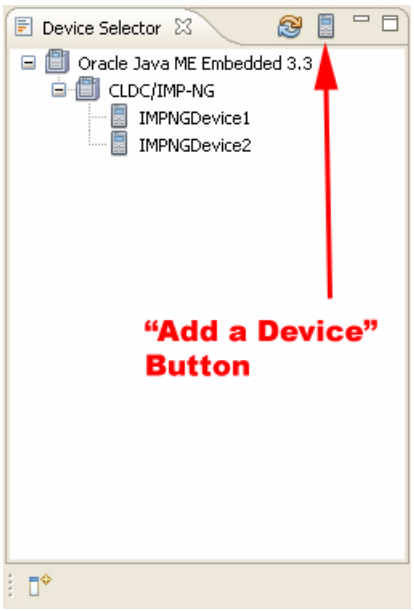
Adding the Keil MCBSTM32F200 Board to the Device Selector

Follow these steps to add the board to the Device Selector in the Oracle Java ME SDK:

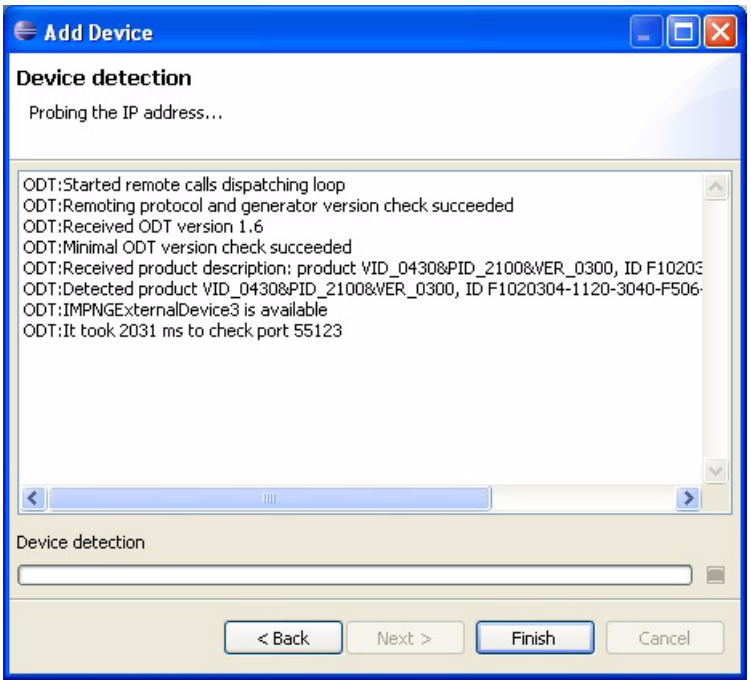
1. Ensure that the property `odt_run_on_start` is set to true in the file `java/jwc_prop.ini` on the Keil MCBSTM32F200.
2. Ensure that the board is receiving power and the Oracle Java ME Embedded software is running.
3. If necessary, open TCP port 55123 in the firewall settings of your computer. The exact procedure to open a port differs depending on your version of Windows or the firewall software that is installed on your computer.
4. Start the Eclipse IDE. In the Eclipse IDE, go to **Window** -> **Show View** -> **Other**. In the popup window that appears, expand the **Java ME** node and select **Device Selector**.



5. On the Device Selector, click on the **Add a Device** button at the top of the Device Selector window.



6. Write the IP address of the Keil MCBSTM32F200 board in the **IP Address** field and click **Next**. You can find the IP address of the Keil MCBSTM32F200 board by looking at the touchscreen on the board.



7. Once the device is detected, click **Finish** on the Add Device screen.

The list of devices in the Device Selector should now include `IMPNGExternalDevice1`.

Assigning the Keil MCBSTM32F200 Board to Your Project

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

1. Right-click on your project and choose **Properties**.
2. Select the **Java ME** category on the properties window.
3. Select **IMPNGExternalDevice1** from the device list. If the device is not shown, add it using the **Add...** button, selecting **Oracle Java ME Embedded 3.3** as the SDK and **IMPNGExternalDevice1** as the device.

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

1. Select **New** -> **Other**. Then expand the **Java ME** tree node, and create a new **MIDlet Project**.
2. Expand the **Java ME** tree node, and create a new **MIDlet Project**.
3. In the Configuration pane of the creation dialog, select **IMPNGExternalDevice1** from the device list.
4. Select the appropriate **Profile** and **Configuration** for your project.

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

Sample Source Code

Once the project is created, use the source code in Example 2–2 for a default source file.

Example 2–2 Sample Code to Access a GPIO Port

```
package embeddedapplication1;

import com.oracle.deviceaccess.PeripheralManager;
import com.oracle.deviceaccess.PeripheralNotAvailableException;
import com.oracle.deviceaccess.PeripheralNotFoundException;
import com.oracle.deviceaccess.gpio.GPIOPin;
import java.io.IOException;
import javax.microedition.midlet.*;


public class IMlet extends MIDlet {

    public void startApp() {

        try {
            GPIOPin pin = (GPIOPin)PeripheralManager.open(2);
            boolean b = pin.getValue();
        } catch (PeripheralNotAvailableException ex) {
            ex.printStackTrace();
        } catch (PeripheralNotFoundException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }

    }

    public void pauseApp() {
    }

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

This sample application will obtain an object representing GPIO pin 1 from the `PeripheralManager`, and attempt to obtain its high/low value.

Accessing the Peripherals on the Keil MCBSTM32F200

There are two ways to allow access to the peripherals on the Keil MCBSTM32F200. The first is to use unsigned applications and modify the security policy file, and the second is to digitally sign the application with the appropriate API permissions requested in the JAD file.

Method #1: Modifying the Security Policy File

Modifying the security policy file is only necessary in the event that a user must manually install the application on the board, at which point the unsigned application will be installed in the `untrusted` security domain.

With this method, simply add the line "`allow: device_access`" to the "`untrusted`" domains of the security policy file. By default, this is located on the SD card in the `appdb/_policy.txt` file, but be sure to check the `security.policy` file entry in the `java/jwc_prop.ini` file to verify the current file name.

Note that if an application is installed on the board using NetBeans or Eclipse during development, the application will automatically be installed in the `maximum` security domain as a convenience. Manual installation, however, will install the unsigned application into the `untrusted` security domain. Note that after development is finished, you should publish your applications with signed API permissions.

Method #2: Signing the Application with API Permissions

The second method is more complex, but is the preferred route for applications that are widely distributed. Open the Application Descriptor for your project in the Packages window, and select the **Application Descriptor** pane. You will need to manually add or change the following lines in the Application Descriptor.

```
MIDlet-Permissions: com.oracle.deviceaccess.gpio
Microedition-Profile: IMP-NG
```

Applications that access the Device Access APIs must also be signed. Here are the instructions on how to setup a keystore with a local certificate that can be used to sign the applications.

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

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

   This command will generate 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.

2. Copy the `appdb/_main.ks` keystore file from the Keil MCBSTM32F200 over to the desktop and 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 3.3 distribution.

```
{mekeytool} -import -MEkeystore _main.ks -keystore mykeystore.ks
-storepass spass -alias mycert -domain trusted
```

This will import the information in `mykeystore.ks` you just created to the `_main.ks` keystore. Once this is completed, copy the `_main.ks` file back to its original location on the Keil MCBSTM32F200.

Use the following steps to sign your application before deploying to the Keil MCBSTM32F200 board.

1. Right click your project and select **Properties**.
2. Choose the **Signing** option under the **Java ME** category.
3. Check the **Enable Project Specific Settings** checkbox. Import the `mykeystore.ks` file that you created as an **External...** keystore file. Provide the keystore and key passwords that you created earlier. Ensure that the **mycert** key alias is present.
4. Ensure that the project is being signed in the project's Application Descriptor. When the project is built and run, it will be digitally signed when deployed to the Keil MCBSTM32F200.

Debugging an IMlet on the Keil MCBSTM32F200 Board

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

1. Open your IMlet class on the Eclipse editor.
2. Click once directly on the line number where you want to set a breakpoint. The line number has a small circle next to it to indicate a breakpoint.
3. Select **Run** -> **Debug** or use the Debug button on the toolbar.

The debugger connects to the debug agent on the board and the program execution stops at your breakpoint, as shown in Figure 2–4.

Figure 2–4 Debugging an IMlet on the Board Using the Eclipse IDE

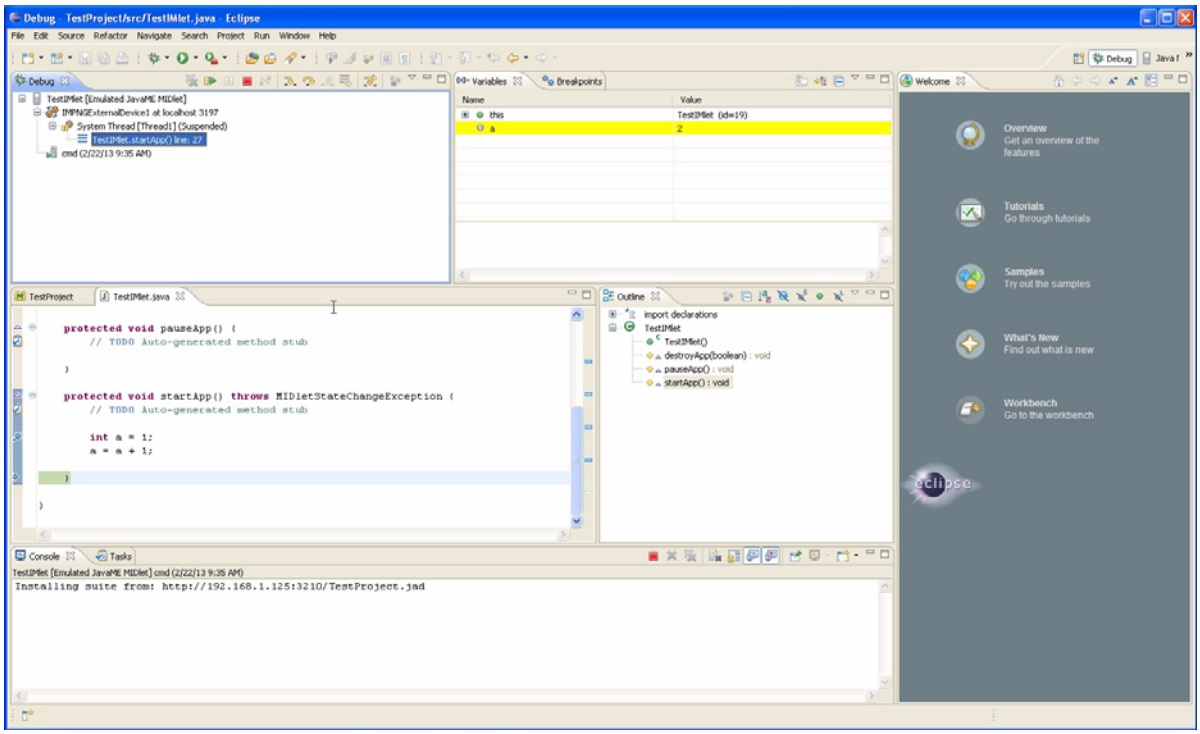


Figure 2–4 shown an entire Eclipse debugging environment that allows the programmer to execute a program step by step as well as add and remove variables from a watch list on the bottom of the screen.

For more information on using the device access APIs, please see the Device Access API Guide and the associated javadocs.

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 Keil MCBSTM32F200 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
- Starting Oracle Java ME Embedded on the Board
- Using the Board with the Oracle Java ME SDK and the NetBeans IDE
- Other Problems

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 `flash.bat` script fails and the firmware is not installed on the board.	The debugger is not connected.	Connect the debug cable to the board and to the computer. Both the power cable and the debug cable must be connected to install the firmware on the board.
(*continued*)	Your debugger is not the ULINK ME debugger.	Follow these steps: 1. Open the file `flash/jmee.uvproj` from the Oracle Java ME Embedded distribution with the Keil µVision IDE. 2. Select the menu **Flash** > **Configure Flash Tools**. 3. Choose your debugger from the list under Use Target Driver for Flash Programming and click **OK**. 4. Select the menu **Flash** > **Download**. The firmware installs on the board.
An output file specified in the `flash.bat` is not generated.	There is a bug in version 4.6 of the MDK-ARM toolkit.	Use MDK-ARM 4.54 or 4.7

Starting 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.
An authorization failure is given when an IMlet attempts to access any Device Access API.	The IMlet is not signed.	Sign the IMlet using a keystore with a trusted certificate authority (CA).
(continued)	The date on the board may invalidate the certificate used to authenticate the digital signature.	See the section Setting Up the MicroSD Card in Chapter 1 for the procedure on how to reset the date on the board by modifying the `rtc_upd.cfg` file.
Oracle Java ME Embedded fails to initialize the network on the board.	The network configuration is incorrect.	Edit the file `platform.cfg` in the SD card and edit the network configuration parameters. Insert the SD card on the board and press the Reset button.
Oracle Java ME Embedded fails to reset the clock on the board.	The `rtc_upd.cfg` configuration file was not renamed prior to board initialization.	Rename the `rtc_upd.cfg` file on the SD card to `rtc.cfg` and reset the board.

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 board is not detected when adding a new device to the Device Selector.	On-device debugging is not enabled.	Edit the file `java/jwc_prop.ini` and set the property `odt_run_on_start` to `true`. Insert the SD card on the board and press the Reset button.
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.

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

Problem	Cause	Solution
The board resets itself, especially when installing large IMlets or debugging applications.	In some cases the watchdog may trigger while the board communicates with the Oracle Java ME SDK and the NetBeans IDE.	Disable the watchdog on the file `platform.cfg`. See the section Setting Up the MicroSD Card in Chapter 1. Insert the SD card on the board and press the Reset button.
The debugging session freezes, disconnects unexpectedly, or shows error messages.	The firewall on the computer is blocking some debugging traffic.	Open TCP port 2808 on your firewall configuration settings. The exact procedure to open a port differs depending on your version of Windows or your firewall software.
(*continued*)	Thunderbird is using a port that is needed for communication with the board.	Close `thunderbird.exe` during the debugging session.

Other Problems

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

If you cannot view the TCP log, the system log of the Keil MCBSTM32F200 board can provide you with the same information. Follow these steps to view the system log:

1. Connect a serial cable from the computer to the board. Use the upper DB9 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.

A

Keil MCBSTM32F200 Board Peripheral List

This appendix describes the proper ID and names for the various peripheral ports and buses for the Keil MCBSTM32F200 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.

The Keil Reference Manual for the MCBSRM32F200 board provides additional technical information about the peripherals accessible via 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
110	ADC1	On-board Potentiometer connected to PF9	converterNumber = 1 channelNumber = 7 resolution = 12 samplingInterval = 14857 samplingTime = 15000 (ignored)

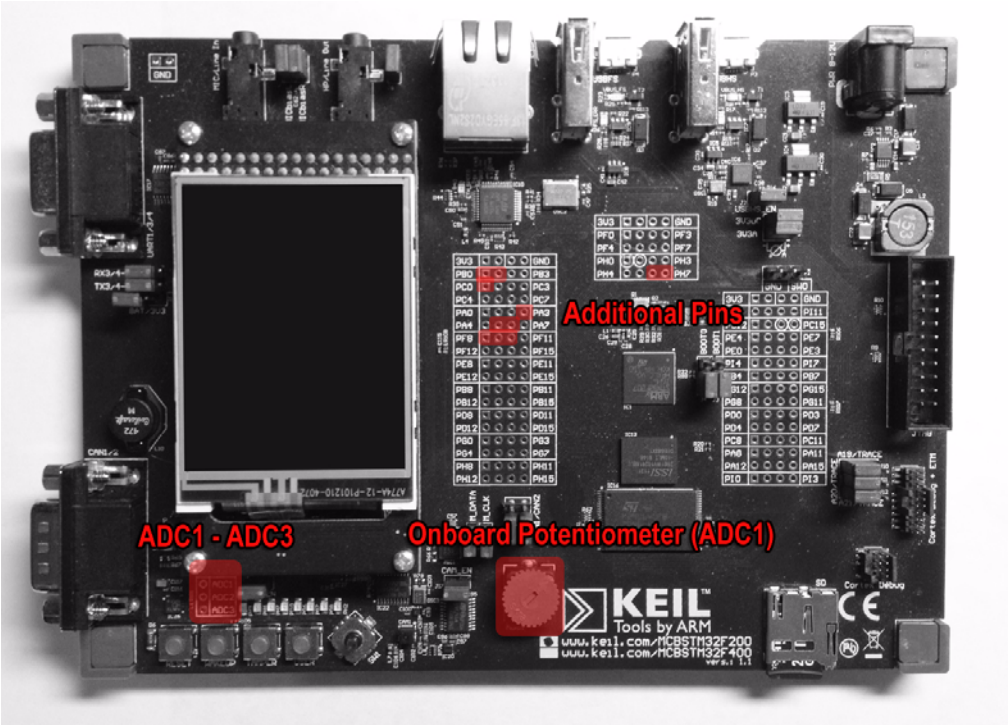
The following additional ADC channels are available ad-hoc.

Mapped To	Converter(s)	Channel	Notes
PF6	2	4	Not connected
PF7	2	5	Not connected
PF8	2	6	Not connected
PF10	2	8	Not connected
PA3	0-2	3	USBHS (USBHS_EN J2)
PA5	0-1	5	Microphone (M_DATA J5), DAC2
PA6	0-1	6	Camera (CAM_EN J17)
PB0	0-1	8	USBHS (USBHS_EN J2)
PB1	0-1	9	USBHS (USBHS_EN J2)

Mapped To	Converter(s)	Channel	Notes
PC0	0-2	11	USBHS (USBHS_EN J2)
PF9	2	7	Potentiometer

Additional notes:

- There are 48 total channels. For more information on assigned external pins, see the manufacturer board schematics.
- For the converter number, acceptable values range from 0 to 2, which map to ADC1 to ADC3 on the device. For the channel number, acceptable values range from 0 to 15. Note that only one channel may be opened on each converter at a time. Consequently, only three channels may be in operation at any one time.
- The internal sampling values range from [14857, 46857]. However, the sampling time is ignored.



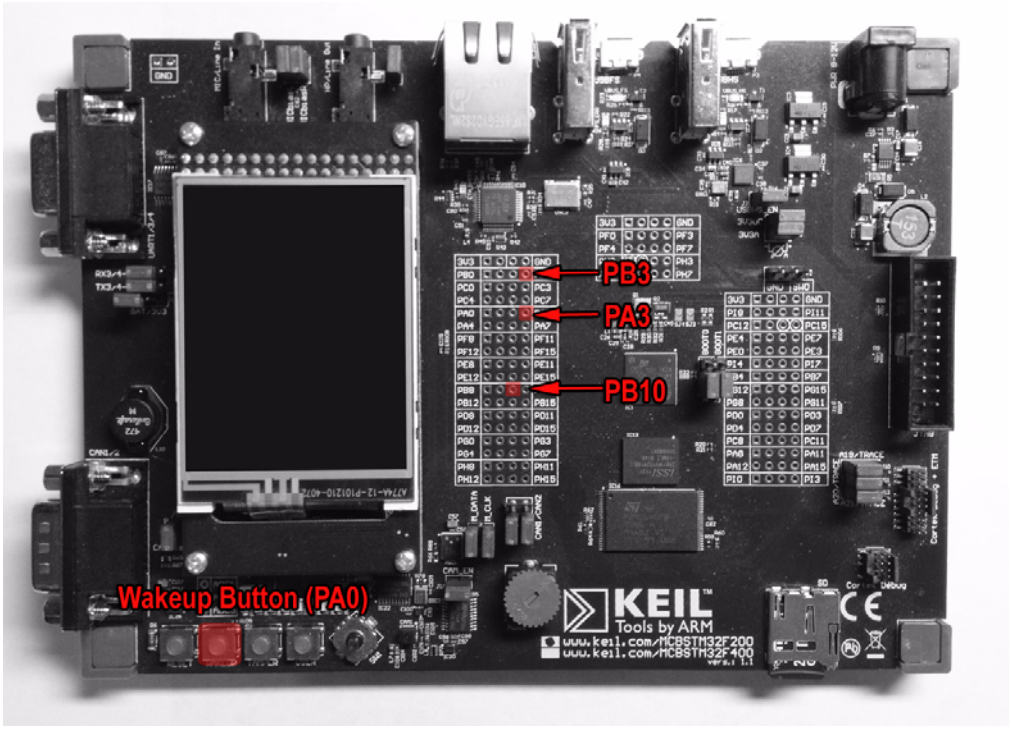
Pulse Counter

There are four pulse counters on the board. Each has a GPIO pin source: PA0, PB3, PB10, and PA3. Note that each source has only one counter, so the "number" property is not used for the Keil board.

The following pulse counters are pre-configured:

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
201	COUNTER_PA0	"WAKEUP" button connected to PA0	counterNumber - not supported / ignored type = PulseCounterConfig.TYPE_FALLING_EDGE_ONLY portNumber = 0 pinNumber = 0
202	COUNTER_PB3	PB3	counterNumber - not supported / ignored type = PulseCounterConfig.TYPE_FALLING_EDGE_ONLY portNumber = 1 pinNumber = 3
203	COUNTER_PB10	PB10	counterNumber - not supported / ignored type = PulseCounterConfig.TYPE_FALLING_EDGE_ONLY portNumber = 1 pinNumber = 10
204	COUNTER_PA3	PA3	counterNumber - not supported / ignored type = PulseCounterConfig.TYPE_FALLING_EDGE_ONLY portNumber = 0 pinNumber = 3

There are no pulse counters available ad-hoc.

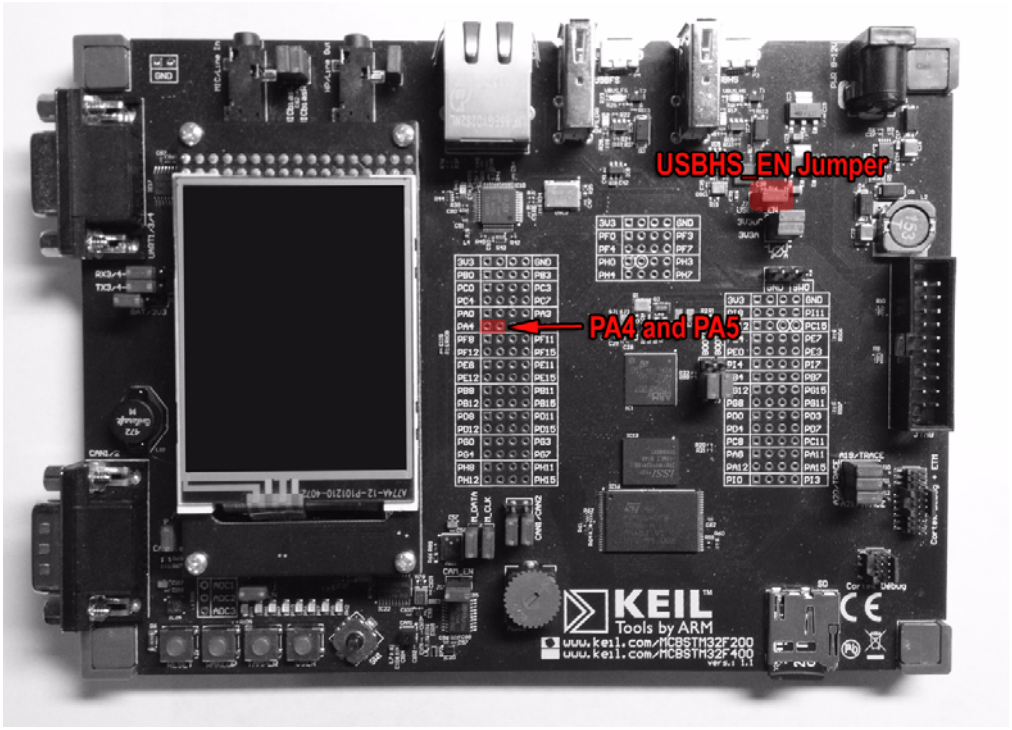


Digital-to-Analog Converter (DAC)

The following DAC devices are pre-configured:

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
120	DAC1	PA4, connected to Analog input 'AINT1A' of CODEC IC11	converterNumber = 0 channelNumber = 0 resolution = 12 samplingInterval = 1000
121	DAC2	PA5, connected to 'CLK' input of HS USB OTG transceiver IC6. To use DAC2 the IC should be disabled by disconnecting J2 'USBHS_EN' jumper	converterNumber = 0 channelNumber = 1 resolution = 12 samplingInterval = 1000

There are no DAC channels available ad-hoc.



GPIO Pins

The following GPIO pins are pre-configured:

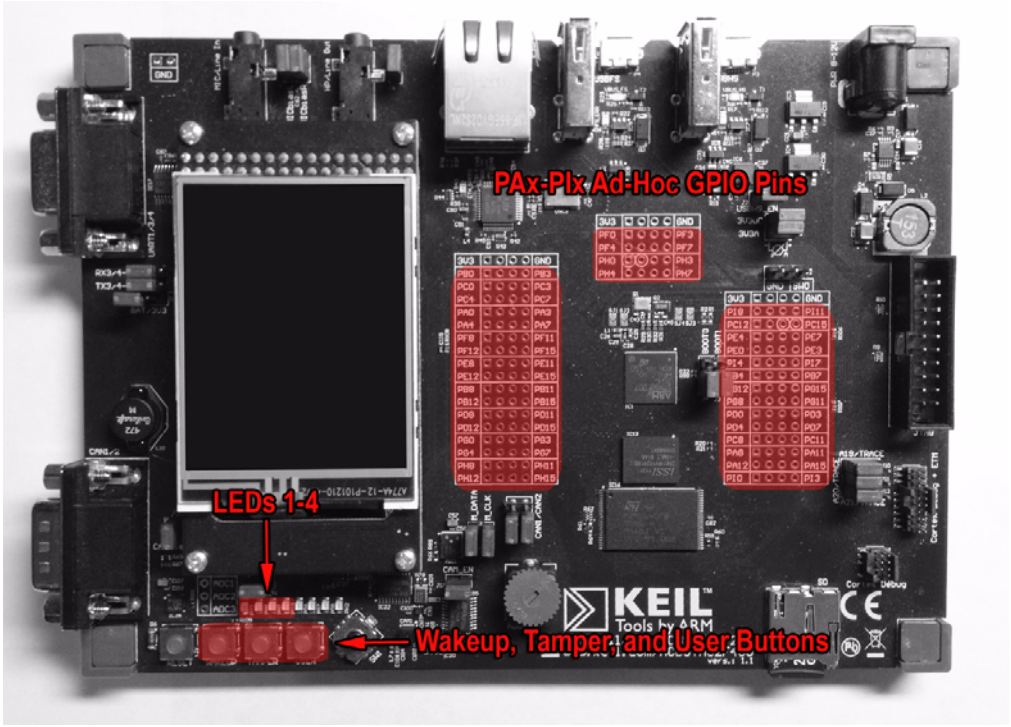
DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
1	LED 1	LED PH3	portNumber = 7 pinNumber = 3 direction = GPIOPinConfig.DIR_OUTPUT_ONLY mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL trigger = GPIOPinConfig.TRIGGER_NONE initValue = false
2	LED 2	LED PH6	portNumber = 7 pinNumber = 6 direction = GPIOPinConfig.DIR_OUTPUT_ONLY mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL trigger = GPIOPinConfig.TRIGGER_NONE initValue = false
3	LED 3	LED PH7	portNumber = 7 pinNumber = 7 direction = GPIOPinConfig.DIR_OUTPUT_ONLY mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL trigger = GPIOPinConfig.TRIGGER_NONE initValue = false
4	LED 4	LED PI10	portNumber = 8 pinNumber = 10 direction = GPIOPinConfig.DIR_OUTPUT_ONLY mode = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL trigger = GPIOPinConfig.TRIGGER_NONE initValue = false
5	WAKEUP_BUTTON or BUTTON 1	"WAKEUP" button	portNumber = 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	TAMPER_BUTTON or BUTTON 2	"TAMPER" button	portNumber = 2 pinNumber = 13 direction = GPIOPinConfig.DIR_INPUT_ONLY mode = GPIOPinConfig.MODE_INPUT_PULL_UP trigger = GPIOPinConfig.TRIGGER_FALLING_EDGE initValue - ignored
7	USER_BUTTON or BUTTON 3	"USER" button	portNumber = 6 pinNumber = 15 direction = GPIOPinConfig.DIR_INPUT_ONLY mode = GPIOPinConfig.MODE_INPUT_PULL_UP trigger = GPIOPinConfig.TRIGGER_FALLING_EDGE initValue - ignored

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.

The following GPIO pins are available ad-hoc. Note that some GPIO pins are used at the system level, so opening those pins may affect board behavior.

- Accepted port numbers are PA=0, PB=1, PC=2, PD=3, PE=4, PF=5, PG=6, PH=7, PI=8.
- Accepted pins number are from 0 to 15.
- Depending on the board, PB0 and PB1 may be unused.



GPIO Ports

The following GPIO ports are pre-configured:

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
8	LEDS	PH3, PH6, PH7, PI10	direction = GPIOPortonfig.DIR_OUTPUT_ONLY initValue = 0 mode of pins = GPIOPinConfig.MODE_OUTPUT_PUSH_PULL

The following GPIO ports are available ad-hoc:

- All combinations of available pins

Inter-Integrated Circuit (I2C)

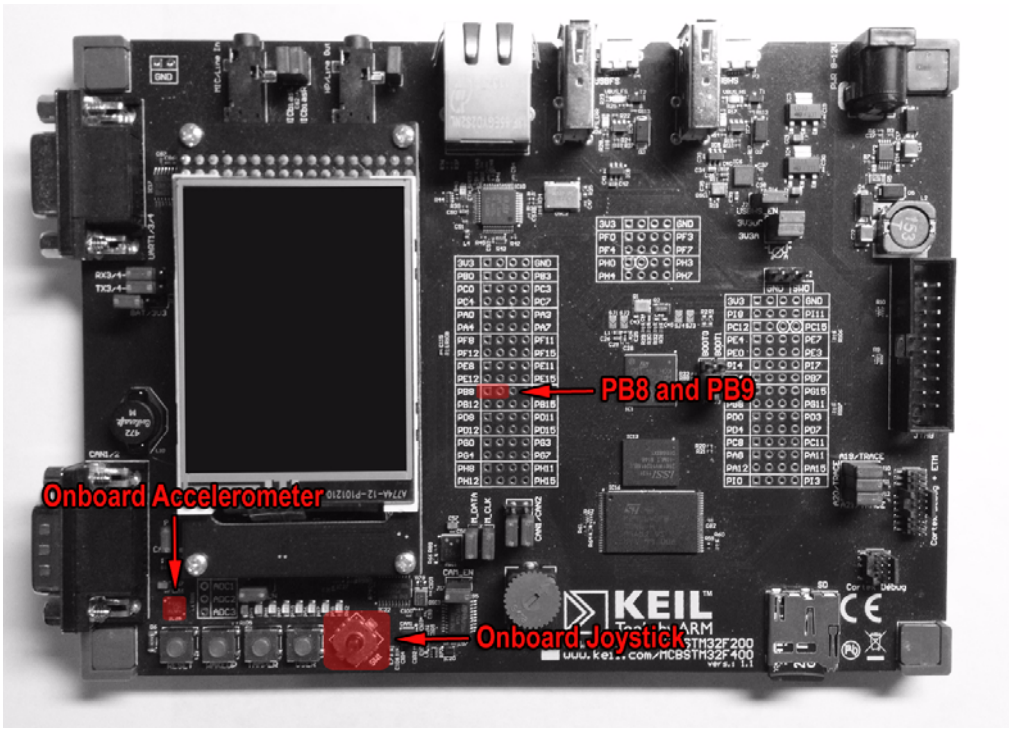
The following I2C devices are pre-configured:

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
50	I2C_JOYSTICK	On-board Joystick connected via I2C bus	busNumber = 0 address = 68 clockFrequency = 100000

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
51	LIS3DH	LIS3DH MEMS motion sensor(On-board 3-axis accelerometer connected via I2C bus)	busNumber = 0 address = 24 clockFrequency = 100000

The following I2C devices are available ad-hoc using connectors PB8 and PB9:

- Any address on bus 0



MMIO

The following MMIO peripherals are available:

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
10	TEST_DEVICE		byteOrdering = Peripheral.LITTLE_ENDIAN
1001	WDOGLOG		byteOrdering = Peripheral.LITTLE_ENDIAN
999	RTC		byteOrdering = Peripheral.LITTLE_ENDIAN

The MMIO raw memories are shown here:

DAAPI Peripheral ID	Name	Address	Type and Size
10	BYTE	0x2001FE00	byte 1
10	SHORT	0x2001FE04	short 2
10	INT	0x2001FE08	int 4
10	BLOCK	0x2001FE20	block 256
1001	WDOG_LOG	0x607F0000	block 65536
999	WPR	0x40002824	int 4
999	CR	0x40002808	int 4
999	DR	0x40002804	int 4
999	TR	0x40002800	int 4
999	ISR	0x4000280C	int 4

Note that the addresses [0x2001FE00 - 20020000) are part of the external RAM reserved for MMIO, and any type will be allowed. In addition, the addresses [0x40000000 - 0x50060C00) and [0xA0000000 - 0xA0001000) are peripheral IO registers. Be sure to check the manufacturer's documentation for actual sizes, type and valid values. With each of the addresses above, the final address is non-inclusive.

There are no devices with event support.

SPI

The following SPI devices are pre-configured:

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
12	SPI2_Slave	MOSI on PB15	busNumber = 1
		MISO on PB14	address = 0
		SCK on PB13	clockFrequency = 2000000
		CS on PB12	clockMode = 1
			wordLength = 8
			bitOrdering = 1
			Configuration is tested with external SPI Digital thermometer (DS1722, Dallas Semiconductor)

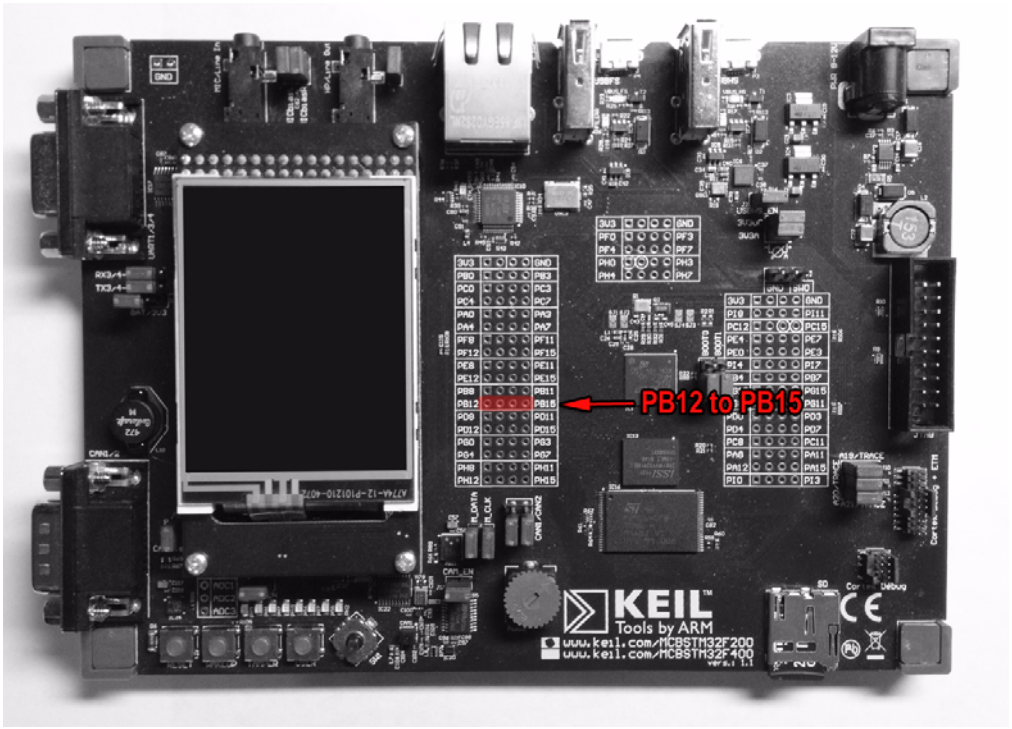
The following SPI devices are available ad-hoc:

- SPI bus numbers: SPI1 = `0`; SPI2 = `1`; SPI3 = `2`

  Clock frequency: 2MHz = `2000000`; 25MHz = `25000000`; 50MHz = `50000000`; 100MHz = `100000000`

  Clock polarity and phase: 0 = `CPOL_Low | CPHA_1Edge`; 1 = `CPOL_Low | CPHA_2Edge`; 2 = `CPOL_High | CPHA_1Edge`; 3 = `CPOL_High | CPHA_2Edge`

Bit ordering for data transfers: MSB = 1; LSB = 0



UART

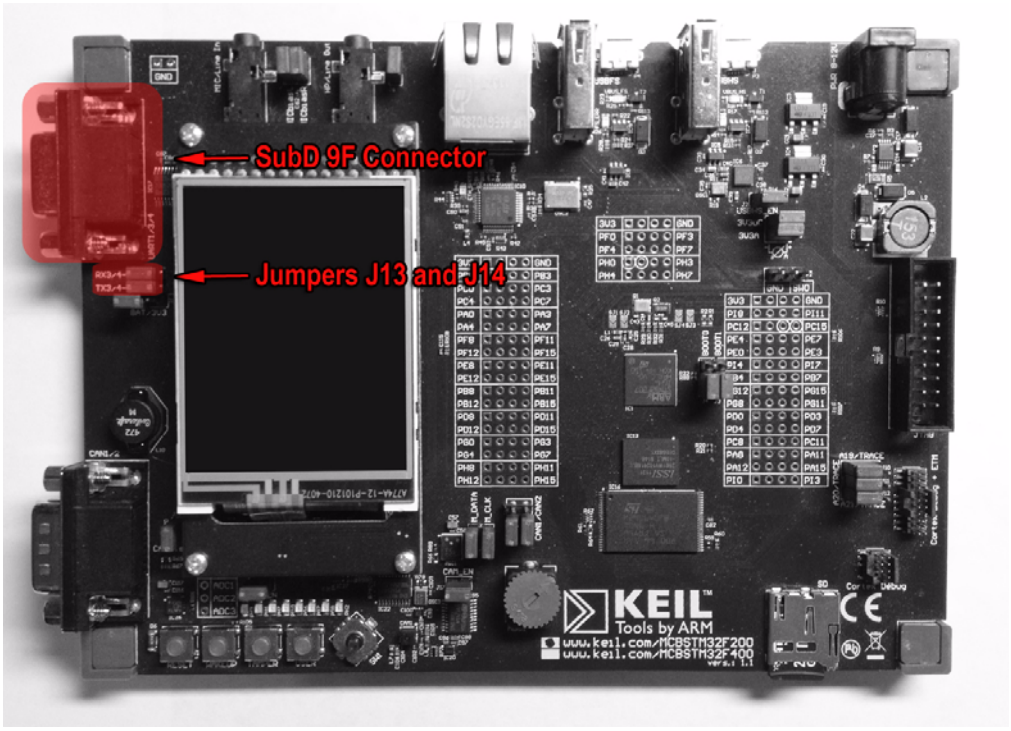
The following UART devices are pre-configured:

DAAPI Peripheral ID	DAAPI Peripheral Name	Mapped To	Configuration
14	UART	Connected to SubD 9F connector if jumpers J13 and J14 are set to TX3/4 and RX3/4 (for 1.1 boards version only)	uartNumber = 0 baudRate = 19200 dataBits = UARTConfig.DATABITS_8 or UARTConfig.DATABITS_9 parity = UARTConfig.PARITY_NONE or UARTConfig.PARITY_EVEN or UARTConfig.PARITY_ODD stopBits = UARTConfig.STOPBITS_1 or UARTConfig.STOPBITS_2 flowcontrol = UARTConfig.FLOWCONTROL_NONE

The are no UART devices available ad-hoc.

The `deviceaccess.uart.prefix` property in the `jwc_prop.ini` file may contain a prefix for easy conversion of the `UARTConfig.portNumber` value to a platform-specific port name. For example, the property may be set to "`COM`" in a Windows environement,

or "UART" in a Keil environment such that appending on a port number will correctly map to the port name.



Watchdog

The following watchdog devices are pre-configured:

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

There are no watchdog devices available ad-hoc.

B

AMS Installer Error Codes

Table B–1 lists the error codes that the AMS command-line interface shows when the installation of an IMlet fails. The description of each code contains more information about the problem that caused the error.

Table B–1 Installer Error Codes

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

Table B–1 (Cont.) Installer Error Codes

Constant	Error Code	Description
CIRCULAR_COMPONENT_DEPENDENCY	64	Circular dynamic component dependency.
COMPONENT_DEPS_LIMIT_EXCEEDED	65	Dynamic component dependencies limit exceeded.
COMPONENT_NAMESPACE_COLLISION	72	The namespace used by a component is the same as another.
CONTENT_HANDLER_CONFLICT	55	The installation of a content handler would conflict with an already installed handler.
CORRUPT_DEPENDENCY_HASH	71	A dependency has a corrupt hash code.
CORRUPT_JAR	36	An entry could not be read from the JAR.
CORRUPT_PROVIDER_CERT	5	The content provider certificate cannot be decoded.
CORRUPT_SIGNATURE	8	The JAR signature cannot be decoded.
DEVICE_INCOMPATIBLE	40	The device does not support either the configuration or profile in the JAD.
DUPLICATED_KEY	88	Duplicated JAD/manifest key attribute
EXPIRED_CA_KEY	12	The certificate authority's public key has expired.
EXPIRED_PROVIDER_CERT	11	The content provider certificate has expired.
INCORRECT_FONT_LOADING	82	A font that is contained with the JAR cannot be loaded.
INSUFFICIENT_STORAGE	30	Not enough storage for this suite to be installed.
INVALID_CONTENT_HANDLER	54	The `MicroEdition-Handler-`<*n*> JAD attribute has invalid values.
INVALID_JAD_TYPE	37	The server did not have a resource with the correct type or the JAD downloaded has the wrong media type.
INVALID_JAD_URL	43	The JAD URL is invalid.
INVALID_JAR_TYPE	38	The server did not have a resource with the correct type or the JAR downloaded has the wrong media type.
INVALID_JAR_URL	44	The JAR URL is invalid.
INVALID_KEY	28	A key for an attribute is not formatted correctly.
INVALID_NATIVE_LIBRARY	85	A native library contained within the JAR cannot be loaded.
INVALID_PACKAGING	87	A dependency cannot be satisfied.
INVALID_PAYMENT_INFO	58	Indicates that the payment information provided with the IMlet suite is incomplete or incorrect.
INVALID_PROVIDER_CERT	7	The signature of the content provider certificate is invalid.

Table B–1 (Cont.) Installer Error Codes

Constant	Error Code	Description
INVALID_RMS_DATA_TYPE	76	The server did not have a resource with the correct type or the JAD downloaded has the wrong media type.
INVALID_RMS_DATA_URL	73	The RMS data file URL is invalid.
INVALID_SERVICE_EXPORT	86	A LIBlet that exports a service with a LIBlet Services attribute does not contain the matching service provider configuration information.
INVALID_SIGNATURE	9	The signature of the JAR is invalid.
INVALID_VALUE	29	A value for an attribute is not formatted correctly.
INVALID_VERSION	16	The format of the version is invalid.
IO_ERROR	102	A low-level hardware error has occurred.
JAD_MOVED	34	The JAD URL for an installed suite is different than the original JAD URL.
JAD_NOT_FOUND	2	The JAD was not found.
JAD_SERVER_NOT_FOUND	1	The server for the JAD was not found.
JAR_CLASSES_VERIFICATION_FAILED	56	Not all classes within JAR package can be successfully verified with class verifier.
JAR_IS_LOCKED	100	Component or MIDlet or IMlet suite is locked by the system.
JAR_NOT_FOUND	20	The JAR was not found at the URL given in the JAD.
JAR_SERVER_NOT_FOUND	19	The server for the JAR was not found at the URL given in the JAD.
JAR_SIZE_MISMATCH	31	The JAR downloaded was not the same size as given in the JAD.
MISSING_CONFIGURATION	41	The configuration is missing from the manifest.
MISSING_DEPENDENCY_HASH	67	A dependency hash code is missing.
MISSING_DEPENDENCY_JAD_URL	66	A dependency JAD URL is missing.
MISSING_JAR_SIZE	21	The JAR size is missing.
MISSING_JAR_URL	18	The URL for the JAR is missing.
MISSING_PROFILE	42	The profile is missing from the manifest.
MISSING_PROVIDER_CERT	4	The content provider certificate is missing.
MISSING_SUITE_NAME	13	The name of MIDlet or IMlet suite is missing.
MISSING_VENDOR	14	The vendor is missing.
MISSING_VERSION	15	The version is missing.
NEW_VERSION	32	This suite is newer that the one currently installed.

Table B–1 (Cont.) Installer Error Codes

Constant	Error Code	Description
NO_ERROR	0	No error.
NOT_YET_VALID_PROVIDER_CERT	89	A certificate is not yet valid.
NOT_YET_VALID_CA_KEY	90	A CA's public key is not yet valid.
OLD_VERSION	17	This suite is older that the one currently installed.
OTHER_ERROR	103	Other errors.
PROXY_AUTH	51	Indicates that the user must first authenticate with the proxy.
PUSH_CLASS_FAILURE	48	The class in a push attribute is not in MIDlet-<*n*> attribute.
PUSH_DUP_FAILURE	45	The connection in a push entry is already taken.
PUSH_FORMAT_FAILURE	46	The format of a push attribute has an invalid format.
PUSH_PROTO_FAILURE	47	The connection in a push attribute is not supported.
REVOKED_CERT	62	The certificate has been revoked.
RMS_DATA_DECRYPT_PASSWORD	83	Indicates that a password is required to decrypt RMS data.
RMS_DATA_ENCRYPT_PASSWORD	84	Indicates that a password is required to encrypt RMS data.
RMS_DATA_NOT_FOUND	75	The RMS data file was not found at the specified URL.
RMS_DATA_SERVER_NOT_FOUND	74	The server for the RMS data file was not found at the specified URL.
RMS_INITIALIZATION_FAILURE	81	Failure to import RMS data.
SUITE_NAME_MISMATCH	25	The MIDlet or IMlet suite name does not match the one in the JAR manifest.
TOO_MANY_PROPS	53	Indicates that either the JAD or manifest has too many properties to fit into memory.
TRUSTED_OVERWRITE_FAILURE	52	Indicates that the user tried to overwrite a trusted suite with an untrusted suite during an update.
UNAUTHORIZED	33	Web server authentication failed or is required.
UNKNOWN_CA	6	The certificate authority (CA) that issued the content provider certificate is unknown.
UNKNOWN_CERT_STATUS	63	The certificate is unknown to OCSP server.
UNSUPPORTED_CERT	10	The content provider certificate has an unsupported version.
UNSUPPORTED_CHAR_ENCODING	61	Indicates that the character encoding specified in the MIME type is not supported.

Table B–1 (Cont.) Installer Error Codes

Constant	Error Code	Description
UNSUPPORTED_PAYMENT_INFO	57	Indicates that the payment information provided with the MIDlet or IMlet suite is incompatible with the current implementation.
UNTRUSTED_PAYMENT_SUITE	59	Indicates that the MIDlet or IMlet suite has payment provisioning information but it is not trusted.
VENDOR_MISMATCH	27	The vendor does not match the one in the JAR manifest.
VERSION_MISMATCH	26	The version does not match the one in the JAR manifest.

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.

RTX

The real-time operating system used on the Keil MCBSTM32F200 embedded platform. The Oracle Java ME Embedded software runs on the Keil 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