Oracle® Java ME Embedded

Developer's Guide

Release 8.3

E73093-02

July 2016

This document is a resource for software developers and release engineers who want to build applications for the Oracle Java ME Embedded software for embedded devices.

ORACLE

Oracle Java ME Embedded Developer's Guide, Release 8.3

E73093-02

Contents

Preface

This book describes how to create and build Oracle Java ME Embedded software from its source code.

Audience

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

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	`$`
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.

Convention	Meaning
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Developer Migration Guide

Learn the changes between version 3.4 of the Oracle Java ME Embedded and the current instance, version 8.3 , if you need to port earlier applications to the latest version of the Oracle Java ME Embedded runtime. You can safely skip this chapter, if you have not developed IMlets using version 3.4 or earlier of the Oracle Java ME Embedded platform.

Topics:

Overview

Java ME 8 is an umbrella terms for two new JSRs: CLDC 8 and MEEP 8. CLDC 8 is a major evolution of CLDC 1.1, while MEEP 8 is a major evolution of IMP-NG. Java ME 8 also includes support for the new Device I/O API.

CLDC 8 is backwards compatible with CLDC 1.1, but includes alignment with the Java SE 7 and 8 language, core APIs, and VM functionality, Java SE-style class-based fine-grain permissions, as well as a significantly enhanced Generic Connection Framework (GCF).

MEEP 8 allows execution of most IMP-NG applications, and includes significant enhancements by leveraging the CLDC 8 features, improvements in the application platform, improved software provisioning and management, footprint scalability through optional APIs, improved connectivity options, and more flexible authentication and authorization mechanisms.

The Device I/O API defines an API that allows Java applications running on small embedded devices to access peripheral devices, from a peripheral device external to the host device to a peripheral chip embedded in the host device.

It is strongly recommended that developers familiarize themselves with the CLDC 8 specification and API, the MEEP 8 specification and API, and the Device I/O API.

Modified Permission Model

There are a number of new permissions that object methods must obtain before they can successfully access peripherals. These permissions are covered in more detail in Chapter 2. However, developers should be aware of the following:

- Java ME 8 now uses Java SE-style class-based fine-grain permissions.

- Applications should request the `jdk.dio.DeviceMgmtPermission` permission when accessing any devices connected to the board through protocols such as GPIO, I2C, SPI, or MMIO, in addition to the permissions required by the communication bus they are using.
- The syntax for the permissions request has changed. The request now includes the device identifier and any specific actions that are requested, if applicable. Device identifiers (e.g., GPIO7, SPI) are listed in the appropriate appendix of the Getting Started Guide for that development board.
- A single request cannot be used for multiple devices; each permissions must be listed separately. For example, you cannot do the following:

  ```
  MIDlet-Permission-1: jdk.dio.GPIOPinPermission "GPIO7,GPIO8" "open"
  ```

  Instead, you must do this:

  ```
  MIDlet-Permission-1: jdk.dio.GPIOPinPermission "GPIO7" "open"
  MIDlet-Permission-2: jdk.dio.GPIOPinPermission "GPIO8" "open"
  ```

  In some cases, you can use an asterisk as a wildcard.

Device I/O Namespace

The Device Access API of the Oracle Java ME Embedded platform is now referred to as the Device I/O API, and is no longer part of the `com.oracle.deviceaccess` package. Instead, all classes now use the `jdk.dio` namespace. In addition:

- Classes that contain "Peripheral" have been changed to "Device." So, for example, `PeripheralManager` has been replaced by `DeviceManager`, and `PeripheralPermission` has been replaced by `DevicePermission`.
- Support now exists for pulse width modulation (PWM) on all platforms.
- Almost all of the individual class methods are unchanged.

Generic Connection Framework Changes

The IMP-NG `javax.microedition` classes are now replaced by the Generic Connection Framework (GCF) with JSR-360 and Java ME Embedded Profile classes (MEEP) with JSR-361. There are a large number of changes that are included in these new profiles. See the specification pages online for more information on each of these classes.

2

Java Embedded VM Proxy and Console

Starting from version 8, the Oracle Java ME Embedded software moves as much CPU intensive processing away from the embedded Java VM as possible. Instead, a separate application running on the host side interacts across the network with the internals of the Java VM. With this design, the VM only sends low-level events to the host application, such as state change information, methods transition, and objects information.

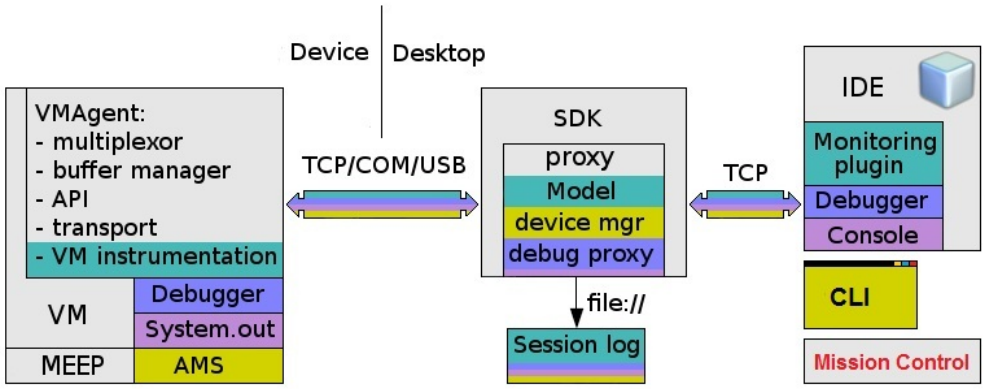
The information is then stored and analyzed on host side, and the host application in turn provides the information to all external profilers, monitors, and managers. External tools can treat the Java SE host application as if it was the VM itself. Besides performance and footprint goals, this approach minimizes development efforts on porting different component communications to new physical transport such as USB, serial, or Bluetooth. Instead, this *VM proxy* application (also known as the Developer Agent) and the VM proxy channel becomes the inter-component tool, and Javacall, CLDC, MEEP, JSRs and SDK components can all take advantage of it.

Topics:

Design

The VM proxy uses a single transport connection to transmit all data for any subsystem. See Figure 2-1 for an illustration of this design; the VM proxy is the middle component.

Figure 2-1 VM Proxy and Agent Design for Java Embedded



Be sure not to confuse the VM proxy with the VM agent. The VM agent consists of native code and is located on the embedded device. The VM proxy is written in Java SE and is launched on the desktop host.

The proxy also provides a software management (SWM) API, similar to the `javax.microedition.swm` package, as declared in the Java ME Embedded Profile (MEEP) specification. This API is an extension of the previous Application Management System (AMS) API of previous versions of the Oracle Java ME Embedded platform, and can be leveraged by ME SDK, IDEs, and the CLI to manage applications with any connected device.

The transport layer between the VM proxy (desktop) and the VM agent (device) is protocol-agnostic by design. However, it is currently implemented for TCP, Serial (COM port), and USB. The transport can initiate connection establishment in any direction, either from device to host or vise versa.

Starting the VM Proxy on the Desktop

To use the VM Proxy, extract the files from your copy of the Oracle Java ME Embedded ZIP archive on the Windows desktop. The VM Proxy program is found as a JAR file inside the `util` directory of the Oracle Java ME Embedded distribution, named `proxy.jar`. You can start the VM Proxy on the desktop host computer either in a server or a client mode as described below.

Server Mode Connection

The server mode is used by default. In this mode, the VM Proxy must be started after the Java runtime is started on the embedded board. Then do the following.

1. 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 -socket <Raspberry Pi IP Address>

Channel 8 CLOSED -> AVAILABLE
Trying to open socket connection with device: <IP Address>:2201
Connected to the socket Socket[addr=/<IP address>, port 2201, localport=54784]
Debugger Connection initialized
```

Client Mode Connection

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

1. Edit the `jwc_properties.ini` file on the embedded 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.
2. Start the Java runtime on the embedded board.
3. Change to the `lib` 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
```

By default, the proxy listens for CLI connections at 65002 port on the host. The port can be changed by passing the `-cliport` option while launching the proxy.

VM Proxy Options

The following options are available when starting the VM Proxy using the **java -jar proxy.jar** command.

no options - runs proxy with default transport. The host opens a server socket and waits for a connection from the embedded device. This means the Java Embedded runtime should be started on the device with its `jwc_properties.ini` file containing the following settings:

```
proxy.connection_mode=client
proxy.client_conncetion_address=(IP address of VM Proxy)
```

-socket <IpAddress> - runs the proxy as a client. This means that the device should open a server socket and wait for a connection from the host. The Java Embedded runtime should be started on the device with its `jwc_properties.ini` file containing the following setting:

```
proxy.connection_mode=server
```

-serial <COM_PORT> – Runs the proxy with a serial transport. This means that the VM proxy communicates with device across the specified serial port.

-debug – Adds additional debugging information when the VM proxy is running.

-i – Runs CLI in the command shell proxy in interactive mode.

-cliAcceptAnyHost – Accepts connections to CLI from any host. By default, only localhost connections are accepted.

-proxyhost <HOST ADDRESS>– Specifies the HTTP proxy address for outbound IP connections from the proxy.

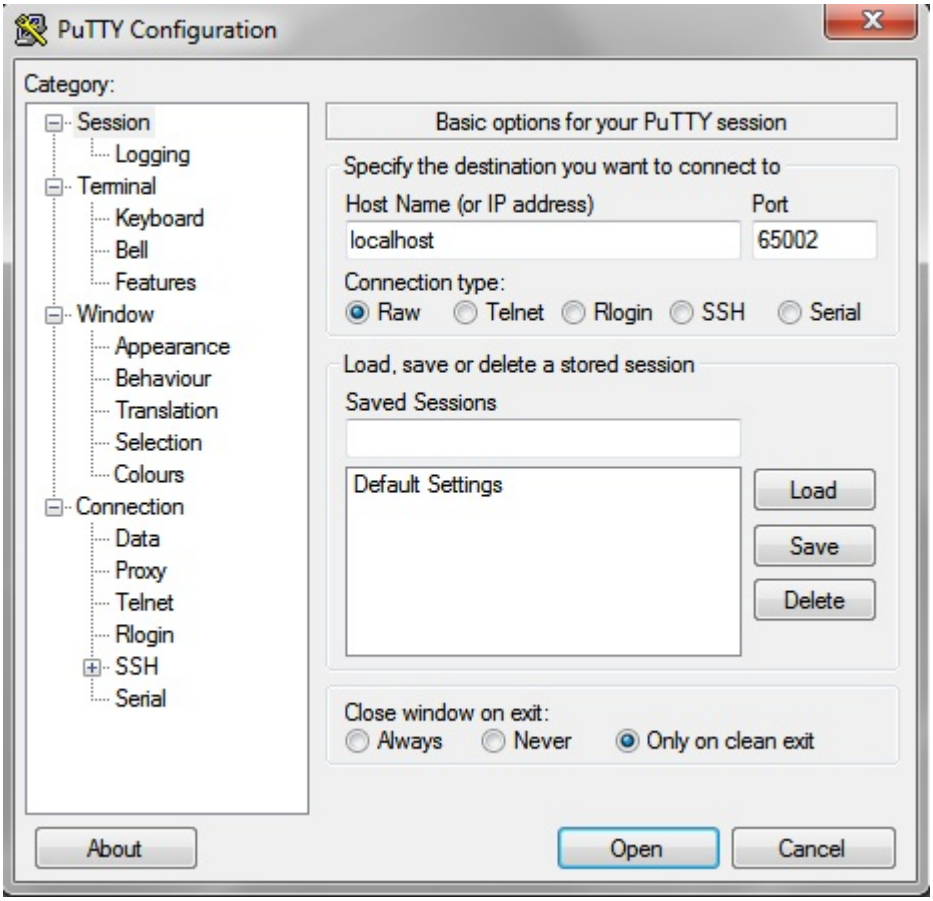
-proxyport <PROXY PORT> – Specifies the HTTP proxy port for outbound IP connections from the proxy.

Use the **-help** proxy option to preview the list of the available options.

Using the Command Line Interface

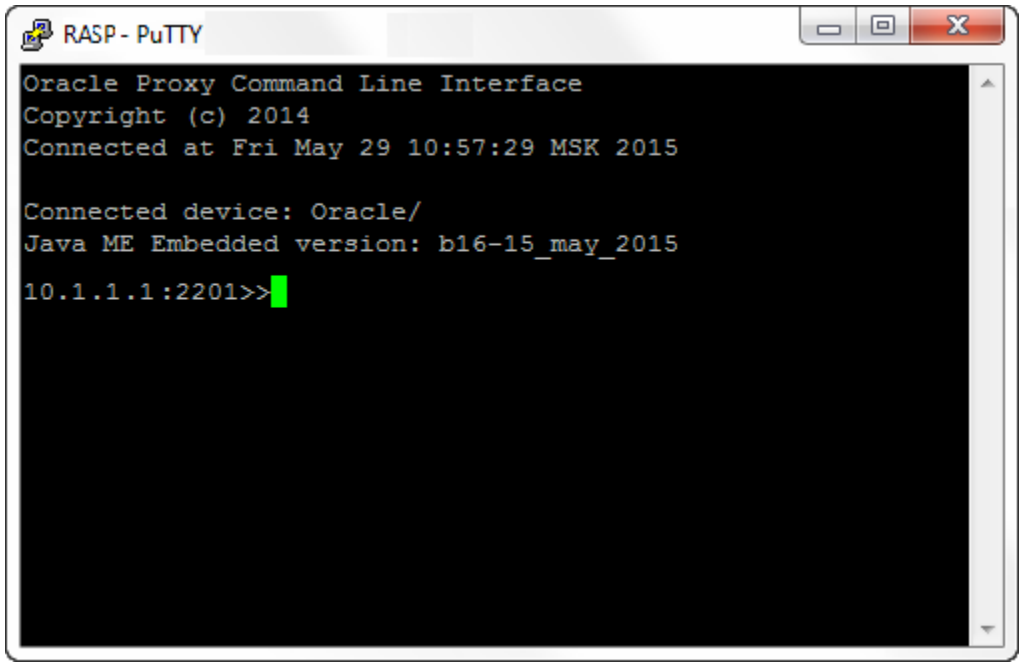
Once the VM proxy is running on the desktop, you can use the AMS CLI. The easiest way to do this is to start a PuTTY executable on your desktop computer, and connect to `localhost` at port 65002. This is shown in Figure 2-1. See the appropriate Getting Started Guide for your embedded board for platform-specific information on using the Command Line Interface.

Figure 2-2 PuTTY Configuration



The window from port 65002 provides a command-line interface (CLI), and is shown in Figure 2-2:

Figure 2-3 Command-Line Interface



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.

The following CLI commands are available for developers. When a command is only available for a specific embedded platform, it is shown in the description.

ams-install

Installs IMlets on the embedded device.

Usage

`ams-install` <*URL*> [`auth=`<*username*>:<*password*>] [`hostdownload`]

Parameters

This command takes the following parameters:

Parameter	Description
<*URL*>	Specifies the JAD/JAR location. The URL may contain credentials to access the JAD/JAR server (e.g. `http://username:password@host/`...).
`hostdownload`	Downloads the JAR file using HTTP and then installs it to device via the tooling channel. Applicable for JAR files only.
`auth`	Specifies the user credentials to access the JAD/JAR server.

Responses
This command may return the following responses:

Response	Description
<<ams-install,start install, <URL>	Information message about the start of the installation process.
<<ams-install, install status: stage <stage> , %percentage%	Information message about the installation progress
<<ams-install, OK,install success	Information message about the installation completing.
<<ams-install,FAIL,missing parameters. see help.	The URL is not specified.
<<ams-install,ERROR,unknown parameter: '<dummy parameter'. see help.	An unexpected parameter was found. One or more parameters were found two or more times.
<<ams-install,FAIL,credentials must be specified once: in url or in auth parameter	Credential info specified twice: in <*URL*> and in <*auth*> parameter.
<<ams-install,FAIL,can't download jar data from <URL>	An error occurred while downloading the JAR in `hostdownload` mode.
<<ams-install,FAIL,errorCode errorcode, errorMessage : message	Installation was aborted for some reason, described in error message.
<<ams-install,FAIL, error occurred exception	An unexpected error occurred. Note that this response is added for debugging purposes and to avoid confusion.

ams-list

Shows a list of installed IMlets on the device or in the specified suite. If no arguments are specified, the `ams-list` command will return a list of all installed suites. If a suite's index or name/vendor combination are used, the command will list the suite's midlets.

Usage
`ams-list` [<*index*> or <*name*>|<*vendor*>]

Parameters
This command takes the following parameters:

Parameter	Description
<*index*>	Specifies the suite via its index number.

Parameter	Description
<name> | *<vendor>*	Specified the suite via its name and vendor

Responses
This command may return the following responses:

Response	Description
<<ams-list,FAIL,invalid parameters	Unexpected parameters were found
<<ams-list,OK,0 suites are installed	No suites were found on the device
<<ams-list,0.name | vendor,status ... **<<ams-list,N.name | vendor,status** **<<ams-list,OK,N suites are installed**	List of installed suites with details
<<ams-list,FAIL,invalid parameter	Parsing the suite's index failed or the | character was missed
<<ams-list,FAIL,not found	The suite was not found
<<ams-list,1.midlet,status ... **<<ams-list,N.midlet,status** **<<ams-list,OK,N midlets are installed in suiteName | suiteVendor**	List of the installed midlets in the suite. Note that each suite status can be RUNNING or STOPPED.

ams-info

Displays information regarding the specified suite.

Usage
`ams-info` *<index>*

Parameters
This command takes the following parameters:

Parameter	Description
<index>	Specifies the suite via its index number.

Responses
This command may return the following responses:

Response	Description
<<ams-info,FAIL,missing parameters. see help.	Number of the specified parameters is less or more than 1.

Response	Description
<<ams-info,FAIL,connection is closed	Connection to the device is closed
<<ams-info,FAIL,not found	The suite is not found or removed during command execution.
<<ams-info,SIZE=<size> **<<ams-info,VERSION=<version>** **<<ams-info,NAME=<name>** **<<ams-info,VENDOR=<vendor>** **<<ams-info,URL=<URL>** **<<ams-info,MIDlet-1=<midlet-1>** **<<ams-info,OK,success getting info**	List of suite-specific information.

ams-update

Updates the specified suite.

Usage
`ams-update` <*index*> *or* <*name* | *vendor*> [`auth`=<*username*>[:<*password*>]]

Parameters
This command takes the following parameters:

Parameter	Description
<*index*>	The index of the suite to be updated. To obtain the suite index, use the `ams-list` command.
<*name*>	<*vendor*>	Specifies the suite to be updated via its name and vendor.
`auth`	Specifies the user credentials to access the JAD/JAR server.

Note: The suite's <*index*> or <*name* | *vendor*> combination is mandatory and must be placed first.

Responses
This command may return the following responses:

Response	Description
<<ams-update,FAIL,missing parameters. see help.	Missing parameters (the suite's index or name | vendor combination is not specified)
<<ams-update,ERROR,unknown parameter: parameter. see help.	An unexpected parameter was found.
<<ams-update,ERROR,duplicate parameter: parameter. see help.	A duplicate parameter was found

Response	Description
<<ams-update,ERROR,Can't update suite suiteIndex (suiteName | suiteVendor): download url is not specified.	The download URL is not specified. For suites, installed in `hostdownload` mode, see the `ams-install` command.
<<ams-update,FAIL,not found	Suite not found. Either the suite was removed or the index / name | vendor identifier was specified incorrectly.
<<ams-update,start install, <URL>	Information message about the update process starting.
<<ams-update, install status: stage stage , percentage%	Information message about the update progress
<<ams-update, OK,install success	Information message about the update process completing.
<<ams-update,FAIL, errorCode errorcode, errorMessage : message	The update was aborted for some reason, as described in the error message.
<<ams-update,FAIL, error occurred exception	An unexpected error occurred. Note that this response is added for debugging purposes and to avoid confusion.

ams-remove

Removes the specified suite from device.

Usage
`ams-remove` <*index or name | vendor*>

Parameters
This command takes the following parameters:

Parameter	Description
<*index*>	The index of the suite to be removed. To obtain the suite index, use the `ams-list` command.
<*name*> | <*vendor*>	Specifies the suite to be removed via its name and vendor

Responses
This command may return the following responses:

Response	Description
<<ams-update,FAIL,missing parameters. see help.	Missing parameters (suite's index or name | vendor not specified)
<<ams-remove,OK,removed	The suite was successfully removed:
<<ams-remove,FAIL,not found	The suite was not found. Either the suite has been already removed, or the <*index*>/<*name* | *vendor*> identifier was specified incorrectly.
<<ams-remove,FAIL,locked	The suite is locked and cannot be removed. The suite is likely in the RUNNING state. The `ams-stop` command must be called first.
<<ams-remove,FAIL,not allowed	The user doesn't have permissions to remove suites.

ams-run

Run default suite's MIDlet or MIDlet, specified wit [MILET_ID] parameter

Usage
`ams-run` <*index* or *name* | *vendor*> [<*id*>]

Parameters
This command takes the following parameters:

Parameter	Description
<*index*>	Index of suite to be run. To obtain the suite index, use the `ams-list` command.
<*name*> | <*vendor*>	Specifies the suite to be launched via its name and vendor
<*id*>	The index of midlet in the suite to be run.

Responses
This command may return the following responses:

Response	Description
<<ams-run,FAIL,invalid parameters	Unexpected parameters were found.
<<ams-run,FAIL,failed to start	Cannot start the midlet. The index of the suite or midlet was specified incorrectly.

Response	Description
<<ams-run,FAIL,already started	The suite has been already started.
<<ams-run,OK,started	The suite was started successfully.

ams-stop

Stops the default MIDlet, or the MIDlet with the specified ID if given.

Usage
`ams-stop` <*index* or *name*|*vendor*> [*id*]

Parameters
This command takes the following parameters:

Parameter	Description
<*index*>	Index of suite to be stopped. To obtain the suite index, use the `ams-list` command.
<*name*>|<*vendor*>	Specified the suite to be stopped via its name and vendor
<*id*>	The ID of midlet in the suite to be stopped.

Responses
This command may return the following responses:

Response	Description
<<ams-stop,FAIL,invalid parameters	Unexpected parameters were found
<<ams-stop,FAIL,not found	Cannot stop the midlet. The index of the suite or midlet was specified incorrectly.
<<ams-stop,OK,started	The suite was stopped successfully

blacklist

Blacklists clients and applications.

Usage
`blacklist -client` <*name*>

`blacklist -app` <*name*|*vendor*>

Parameters
This command takes the following parameters:

Parameter	Description
<*name*>	The name of the client to be blacklist.
<*name*> | <*vendor*>	Specifies the suite to be blacklisted via its name and vendor

Responses
This command may return the following responses:

Response	Description
<<blacklist,FAIL,invalid parameters	Unexpected parameters were found
<<blacklist status OK	The command was successful.

properties-list

Shows the list of names of properties which control Java ME runtime, common to the `java_properties.ini` file. Note that a property type may be only INT, STRING or BOOL. The read/write flag value may be only read/write or read only, and a BOOL property value may be only true or false.

Usage
`properties-list [-l]`

Parameters
This command takes the following parameters:

Parameter	Description
`-l`	Use the long listing format with properties' types, values and readonly flags.

Responses
This command may return the following responses:

Response	Description
<<properties-list,AMS_MEMORY_LIMIT_MVM AMS_MEMORY_RESERVED_MVM AuthenticationName AuthenticationPwd btgoep btl2cap btspp cbs ...	The response without the long listing flag. Shows property names separated by a space.

Response	Description
<<properties-list,OK read/write INT AMS_MEMORY_LIMIT_MVM = -1 read/write INT AMS_MEMORY_RESERVED_MVM = 100 read/write STRING AuthenticationName = user read/write STRING AuthenticationPwd = password read only BOOL microedition.deviceid.isunique = false read only BOOL microedition.devicevendor.isunique = false	The response with the long listing flag.
<<properties-list,FAIL,invalid parameters <<properties-list,Usage: properties-list [-l] <<properties-list,list of properties which control Java ME runtime <<properties-list, -l use a long listing format	An unexpected parameter was found
<<properties-list,OK,there is no property found	An empty list of properties was found.
<<properties-list,FAIL,connection is closed	An `IOException` has occurred.

get-property

Shows the value of requested property. If the property is not defined, the command shows an empty string as its value.

Usage

`get-property` <*name*> `[-i]`

Parameters

This command takes the following parameters:

Parameter	Description
name	The property name
`-i`	Displays additional property information

Responses

This command may return the following responses:

Response	Description
<<get-property,OK,dio.counter = true	The property was found (without displaying additional information)

Response	Description
<<get-property,OK,dummy.property =	The property value is empty or not a set (without additional information)
<<get-property,OK, readonly BOOL dio.counter = true	The property is found (with `-i` flag)
<<get-property,OK, read/write STRING dummy-property =	The property value is empty or not a set (with `-i` flag)
<<get-property,FAIL,illegal argument [ip.netmask] **<<get-property,Usage: get-property name [-i]** **<<get-property,shows value of string property 'name'** **<<get-property, -i display property info**	An unexpected parameter was found The wrong flag format was used(e.g. using `-info` instead of `-i`)
<<get-property,FAIL,connection is closed	An `IOException` has occurred.
<<get-property,OK,LC_NETWORK = 2 => 1	The property was modified with the `set-property` command. The new value will be effective after you restart the device.

set-property

Sets the new value for the requested property. If the property controls the Java ME Runtime (i.e., it is defined in the `java_properties.ini` file), it cannot be rewritten unless the `read-only` flag is disabled. Note that properties are verified for type correctness. The value of a BOOL property may be any string. However, only "`true`" (case insensitive) is considered a true value; any other string is considered to be false.

The new value for a property that controls the Java ME Runtime will be applied only after a VM reboot. In this case, only the latest `set-property` command will have an effect after reboot. New values for other properties can be read just after the `get-property` command has finished.

Usage
`set-property` *<name>* *<value>*

Parameters
This command takes the following parameters:

Parameter	Description
<name>	The name of the requested property
<value>	The new value for the property.

Responses
This command may return the following responses:

Response	Description
<<set-property,OK,imc = new.value	The operation completed successfully.
<<set-property,FAIL,illegal number [hello].	The value type is not a number when property type is INT:
<<set-property,FAIL,illegal argument [microedition.devicevendor.isunique] or [true].	The property is read-only:
<<set-property,FAIL,invalid parameters.	Wrong number of parameters:
<<set-property,FAIL,connection is closed	An `IOException` has occurred

When you modify a property by using the `set-property` command and verify the change with the `get-property` command, the CLI response contains both old and new values as shown in the following example. The new value will be effective after you restart the device.

```
10.162.80.150:2201>>get-property LC_NETWORK
<<get-property,OK,LC_NETWORK = 2
10.162.80.150:2201>>set-property LC_NETWORK 1
<<set-property,OK,LC_NETWORK = 1
Changes will take effect after device restart
10.162.80.150:2201>>get-property LC_NETWORK
<<get-property,OK,LC_NETWORK = 2 => 1
```

save-properties

Saves properties to an internal storage.

Usage

`save-properties`

Parameters

This command takes no parameters:

Responses

This command may return the following responses:

Response	Description
<<save-properties,OK,success	Properties have been successfully saved to the internal storage
<<save-properties,FAIL	An `IOException` has occurred.

net-info

Show the network information of the system. This command only works on Qualcomm IoE devices.

Usage
`net-info`

Parameters
This command takes no parameters:

Responses
This command may return the following responses:

Response	Description
<<net-info,OK,success getting info	Shows network information in the format <*name*>=<*value*>
<<net-info,FAIL, connection is closed	An `IOException` has occurred.

net-set

Sets a new value for the requested property of the network system. The property is verified for type correctness. This command only works on Qualcomm IoE devices.

Usage
`net-set` <*name*> <*value*>

Parameters
This command takes the following parameters:

Parameter	Description
<*name*>	The name of the requested property
<*value*>	The new value for the property.

Responses
This command may return the following responses:

Response	Description
<<net-set,OK,<NAME> = <VALUE>	The operation completed successfully.

Response	Description
<<net-set,FAIL,illegal first argument [<NAME>] **<<net-set ssid <SSID>:set value for WIFI access** **<<net-set passwd <PASSWD>:set password for WIFI access** **<<net-set pref <0|1|2|3|4|5>:set network mode preference 0:AUTO, 1:NO OP, 2:WLAN Only, 3:GSM/WCDMA only, 4:WCDMA only, 5:GSM/WCDMA/WLAN** **<<net-set apn <APN>:set APN** **<<net-set pdp_authtype <0|1|2>:set APN's auth type 0:NONE, 1:PAP, 2:CHAP** **<<net-set pdp_username <USERNAME>:set pdp username** **<<net-set pdp_password <PASSWORD>:set pdp password**	An illegal type of property was encountered. The response dictates the correct syntax and property type.
<<net-set,FAIL,illegal value [<VALUE>]	The value type was not a number when the property type is INT.
<<net-set,FAIL,illegal argument [<NAME>] or [<VALUE>]	This is returned if any of arguments are `null` or if the `property.name` has an incorrect property type.
<<net-set,FAIL,connection is closed	An `IOException` has occurred.

net-reconnect

Reconnects the network and reboots Java.

Usage
`net-reconnect`

Parameters
This command takes no parameters:

Responses
This command may return the following responses:

Response	Description
<<net-reconnect,OK,VM will reboot. Device will reconnect to the network	The network reconnect command completed successfully. The device will be rebooted and reconnected to the network.
<<net-reconnect,FAIL	Cannot reconnect the device to the network

Response	Description
<<net-reconnect,FAIL, connection is closed	An `IOException` has occurred.

device-list

Prints a list of all connected devices at the current time.

Usage
`device-list`

Parameters
This command takes no parameters.

Responses
This command may return the following responses:

Response	Description
< <<device-list, 0,<IP0>:<port0>,CURRENT<<device-list, 1,<IP1>:<port1>...<<device-list,<N-1>,<IPN-1>:<portN-1><<device-list,OK,N devices are connected	Printed list of devices. The "`CURRENT`" annotation indicates the currently seleted device that all device-related CLI command are addressed to.
<<device-list,FAIL,invalid parameters	Unexpected parameters were found. In this case, the command has no parameters, but the user has specified some:

device-change

Switches the currently-selected device. Once changed, all further device-related commands will be address to the newly selected device.

Usage
`device-change` <*index*>

Parameters
This command takes the following parameters:

Parameter	Description
`<index>`	An integer index of device, as printed by the `device-list` command.

Responses
This command may return the following responses:

Response	Description
<<device-change,OK,current device is changed	The command has been processed successfully; the current device was changed.
<<device-change,FAIL,invalid parameters	An invalid number of parameters have been specified (either no parameters or more than one parameter).
<<device-change,FAIL,incorrect device index	The index is not an integer.
<<device-change,FAIL,device not found	There is no such device.
<<device-change,FAIL,the device is already current	An attempt was made to switch to a device that is already the current device.

shutdown

Shutdown or restart the device.

Usage
`shutdown [-r]`

Parameters
This command takes the following parameters:

Parameter	Description
`-r`	Restart the device. Note that restart is not supported on Win32 platform.

Responses
This command may return the following responses:

Response	Description
<<shutdown,OK,device will shutdown!	The `shutdown` command was processed successfully. The device will be shutdown soon.
<<shutdown,OK,device will reboot!	The `shutdown` command was processed successfully, device will be restarted soon.
<<shutdown,FAIL,can't reboot device	Cannot restart the device
<<shutdown,FAIL,wrong parameters. see help.	Unexpected parameters were found.

Response	Description
<<shutdown,FAIL,<Error message>	Shutdown command failed due an unknown reason.

cd

Changes the working directory on the device.

Usage
`cd` <*deviceDirectoryName*>

Parameters
This command takes the following parameters:

Parameter	Description
<*deviceDirectoryName*>	This specifies the directory on the device to which you want to change. The <*deviceDirectoryName*> can be relative to the current working directory, or an absolute path

Responses
This command may return the following responses:

Response	Description
<<cd,OK	The command completed successfully
<<cd,FAIL,invalid parameters	Missing or excess parameters were encountered
<<cd,FAIL,directory not found <deviceDirectoryName>	Incorrect <*deviceDirectoryName*> specified
<<cd,FAIL,connection is closed	An `IOException` has occurred

delete

Deletes file on the device.

Usage
`delete` <*deviceFileName*>

Parameters
This command takes the following parameters:

Parameter	Description
<deviceFileName>	Specifies the file to delete. *<deviceFileName>* can be relative to the current working directory, or an absolute path.

Responses
This command may return the following responses:

Response	Description
`<<delete,OK`	The command completed successfully
`<<delete,FAIL,invalid parameters`	Missing or excess parameters were encountered.
`<<delete,FAIL,file not found <deviceFileName>`	Incorrect *<deviceFileName>* specified
`<<delete,FAIL,connection is closed`	An `IOException` has occurred

get

Copies a device file to the host.

Usage
`get` *<deviceFileName> <hostFileName>*

Parameters
This command takes the following parameters:

Parameter	Description
<deviceFileName>	Specifies the file to copy. *<deviceFileName>* can be relative to the current working directory, or an absolute path.
<hostFileName>	Specifies the name of the file to use on the host.

Responses
This command may return the following responses:

Response	Description
<<get,OK	The command completed successfully
<<get,FAIL,invalid parameters	Missing or excess parameters were encountered
<<get,FAIL,file not found <deviceFileName>	Incorrect *<deviceFileName>* specified

Response	Description
<<get,FAIL,unable to write into file <hostFileName>	Incorrect <*hostFileName*> specified
<<get,FAIL,connection is closed	An `IOException` has occurred

ls

Displays a list of files and subdirectories in a device directory.

Usage
`ls` [<*deviceDirectoryName*>]

Parameters
This command takes the following parameters:

Parameter	Description
<*deviceDirectoryName*>	Specifies the directory for which you want to see a listing. <*deviceDirectoryName*> can be relative to the current working directory, or an absolute path. If no directory is specified, the current working directory on the device is used. In the result listing, subdirectories are marked by a trailing device file separator symbol (for example, "\" on Windows, "/" on RPi).

Responses
This command may return the following responses:

Response	Description
<<ls,OK **alljavalist.txt** **all_classes.zip** **appdb** **bin** **classes** **classes.zip**	The command completed successfully
<<ls,FAIL,invalid parameters	Excess or invalid parameters were encountered
<<ls,FAIL,directory not found <deviceDirectoryName>	Incorrect <*deviceDirectoryName*> specified

mkdir

Creates a directory on the device.

Usage
mkdir <*deviceDirectoryName*>

Parameters
This command takes the following parameters:

Parameter	Description
<*deviceDirectoryName*>	Specifies the name of the new device directory. <*deviceDirectoryName*> can be relative to the current working directory, or an absolute path.

Responses
This command may return the following responses:

Response	Description
<<mkdir,OK	The command completed successfully
<<mkdir,FAIL,invalid parameters	Missing or excess parameters were encountered
<<mkdir,FAIL,directory not found <deviceDirectoryName>	Incorrect <*deviceDirectoryName*> was specified
<<mkdir,FAIL,connection is closed	An `IOException` has occurred

rmdir

Deletes an empty directory on the device.

Usage
rmdir <*deviceDirectoryName*>

Parameters
This command takes the following parameters:

Parameter	Description
<*deviceDirectoryName*>	Specifies the name of the device directory to delete. <*deviceDirectoryName*> can be relative to the current working directory, or an absolute path.

Responses
This command may return the following responses:

Response	Description
<<mkdir,OK	The command completed successfully

Response	Description
<<rmdir,FAIL,invalid parameters>	Missing or excess parameters were encountered
<<rmdir,FAIL,directory not found <deviceDirectoryName>	Incorrect *<deviceDirectoryName>* was specified
<<rmdir,FAIL,unable to delete directory <deviceDirectoryName>	Unable to delete the directory, for example, the directory or a file in it is used by another application, or the directory is not empty.

pwd

Prints the current working directory on the device.

Usage
`pwd`

Responses
This command may return the following responses:

Response	Description
<<pwd,OK c:\Users\abc\javame-sdk\8.0_ea\work \EmbeddedDevice1\appdb	The command processed successfully
<<pwd,FAIL,invalid parameters	Excess parameters were encountered

put

Copies a local host file to the device.

Usage
`put` *<hostFileName>* *<deviceFileName>*

Parameters
This command takes the following parameters:

Parameter	Description
<hostFileName>	Specifies the local host file to copy.
<deviceFileName>	Specifies the name to use on the device. *<deviceFileName>* can be relative to the current working directory, or an absolute path.

Responses
This command may return the following responses:

Response	Description
<<put,OK	The command processed successfully
<<put,FAIL,invalid parameters	Missing or excess parameters
<<put,FAIL,unable to read file <hostFileName>	Incorrect <*hostFileName*> specified
<<put,FAIL,file not found <deviceFileName>	Incorrect <*deviceFileName*> specified
<<put,FAIL,connection is closed	An `IOException` has occurred

exit

Terminates the current CLI session. The CLI server continues to run and the user can re-connect again.

Usage
`exit`

Parameters
The command has no parameters.

Responses
The command returns no response; the terminal application closes.

ks-delete

Deletes a key from the ME device keystore, identified either by its owner or the key number.

Usage
`ks-delete` (`-owner` <*ownerName*> | `-number` <*keyNumber*>) [(`-client` <*name*> | `-proxy`)]

Parameters
This command takes the following parameters:

Parameter	Description
<*ownerName*>	The name of the owner of an ME key.
<*keyNumber*>	The key number (starting at 1) of an ME key currently in the device keystore.
<*name*>	The name of the target security client.
-proxy	Drop the proxy connection security in the next session.

Responses
This command may return the following responses:

Response	Description
<<ks-delete,OK	The command processed successfully
<<ks-delete,FAIL,bad command or missing parameters. Type help for assistance.	An error occurred, either with the command itself or one of the parameters.

ks-export

Exports a key from the device keystore identified by its index.

Usage
`ks-export -number` <*keyNumber*> `-out` <*filename*> `[-client` <*name*>`]`

Parameters
This command takes the following parameters:

Parameter	Description
<*keyNumber*>	The key number (starting at 1) of an ME key currently in the device keystore.
<*filename*>	The complete filename to save the exported key as.
<*name*>	The name of the target security client.

Responses
This command may return the following responses:

Response	Description
<<ks-export,OK	The command processed successfully
<<ks-export,FAIL,bad command or missing parameters. Type help for assistance.	An error occurred, either with the command itself or one of the parameters.

ks-import

Imports a public key from a JCE keystore or a key file into a ME device keystore.

Usage
`ks-import [-keystore` <*filename*>`] [-storepass` <*storepass*>`] [-keypass` <*keypass*>`] [-alias` <*keyAlias*>`] [-client` <*ClientName*>`] [-proxy]`

Parameters
This command takes the following parameters:

Parameter	Description
<filename>	The complete filename of the JCE keystore or the key file. The keystore file may be in the following formats: *jks*, *pkcs12*,*pem*,*der*
<storepass>	The password for the JCA keystore This parameter is not required when the source file is in the *pem* or *der* format.
<keypass>	The password for the private key in a JCA or PKCS12 keystore. This parameter is not required if the command is importing only a public certificate.
<keyAlias>	The short string ID of a key in a JCA keystore. This parameter is not required when the source file is in the *pem* or *der* format.
<clientName>	The name of the security domain client. If specified, the certificate will be added to the indicated client only.
-proxy	Enforces the proxy connection security (TLS 1.2) with the given certificate, starting from the next session.

Responses
This command may return the following responses:

Response	Description
<<ks-import,OK	The command processed successfully
<<ks-import,FAIL,bad command or missing parameters. Type help for assistance.	An error occurred, either with the command itself or one of the parameters.

ks-list

Lists the owner and validity period of each key in the ME device keystore.

Usage
```
ks-list [(-proxy | -client <name>)]
```

Parameters
This command takes the following parameters:

Parameter	Description
<name>	The name of the target security client.
-proxy	Displays the certificate(s) installed for securing the proxy connection.

Responses
This command may return the following responses:

Response	Description
<<ks-list [<*number*>**]=Owner**:<*Owner distinguished name*> **Valid from** <*date*> **to** <*date*>	A list of each of the installed keys, following this format.

ks-clients

Presents a list of all the security clients defined in the system that can accept public keys.

Usage
```
ks-clients
```

Responses
This command may return the following responses:

Response	Description
<<ks-list [<*number*>**]=**<*Client name*>	A list of each of the clients, following this format.

dumpheap

Dumps Java heap of the connected device.

Usage
```
dumpheap [-gc]
```

Parameters
This command takes the following parameters:

Parameter	Description
[-gc]	Forces full garbage collector (GC) on the device prior the dump.

3

Security

This chapter discusses security with the Oracle Java ME Embedded environment. Note that with version 8 of the OJMEE, the security system was changed considerably, and now uses Java SE-style fine-grain permissions. In addition, a security policy must be chosen and JAR files, if applicable, must be digitally signed in order for peripherals to be accessed.

Overview of Oracle Java ME Embedded Permissions

Applications that require access to peripherals or resources must request appropriate permissions in the JAD file. For more information on using the Device I/O APIs, please see the *Device I/O API 1.1* specification and the associated Javadocs at the following site:

`http://docs.oracle.com/javame`

Table 3-1 gives a list of all permissions that can be requested in the Oracle Java ME Embedded environment, as well as a description of when they are applicable.

Table 3-1 Oracle Java ME Embedded Permissions

Permission	Description
`com.oracle.crypto.keystore.KeyStorePermission`	Allows access to the keystore
`com.oracle.ssl.DTLSServerPermission`	Allows access to a DTLS server.
`com.oracle.runtime.update.RuntimeUpdatePermission`	Allows remote update of the Java ME Embedded runtime.
`java.io.FilePermission`	Accessing files
`java.lang.RuntimePermission`	Accessing runtime properties
`java.util.logging.LoggingPermission`	Use of log files
`java.util.PropertyPermission`	Accessing system properties
`javax.microedition.apdu.APDUPermission`	Access to smartcards using the APDU protocol
`javax.microedition.cellular.CellularPermission`	Use of cellular telephone functionality on a board.
`javax.microedition.event.EventPermission`	Reading and posting system-level events
`javax.microedition.io.AccessPointPermission`	Use of access points for network connections.

Table 3-1 (Cont.) Oracle Java ME Embedded Permissions

Permission	Description
`javax.microedition.io.CommProtocolPermission`	Use of the COMM serial port protocol
`javax.microedition.io.Connector.cbs`	Use of a Cell Broadcast Service (CBS) Connector
`javax.microedition.io.Connector.file.read`	Use of a file read Connector
`javax.microedition.io.Connector.file.write`	Use of a file write Connector
`javax.microedition.io.Connector.rtsp`	Use of a real-time streaming protocol (RTSP) Connector
`javax.microedition.io.Connector.sms`	Use of an SMS Connector
`javax.microedition.io.DatagramProtocolPermission`	Use of the datagram protocol
`javax.microedition.io.DTLSProtocolPermission`	Use of the Datagram Transport Layer Security (DTLS) protocol
`javax.microedition.io.FileProtocolPermission`	Use of a file protocol
`javax.microedition.io.HttpProtocolPermission`	Use of the HTTP protocol
`javax.microedition.io.HttpsProtocolPermission`	Use of the HTTPS protocol
`javax.microedition.io.IMCProtocolPermission`	Use of the Inter-MIDlet communication protocol
`javax.microedition.io.MulticastProtocolPermission`	Use of a multicast protocol
`javax.microedition.io.PushRegistryPermission`	Use of a push registry
`javax.microedition.io.SocketProtocolPermission`	Use of a socket protocol
`javax.microedition.io.SSLProtocolPermission`	Use of the Secure Sockets Layer (SSL) protocol
`javax.microedition.location.LocationPermission`	Obtain the current location
`javax.microedition.media.control.RecordControl`	Use of a recording feature on the device
`javax.microedition.media.control.VideoControl.getSnapshot`	Use of a video snapshot feature on the device

Table 3-1 (Cont.) Oracle Java ME Embedded Permissions

Permission	Description
`javax.microedition.midlet.AutoStartPermission`	A permission to autostart an IMlet suite on a device
`javax.microedition.power.PowerStatePermission`	Access the current power state of the device
`javax.microedition.swm.SWMPermission`	Access the software management features of the Java ME Embedded runtime
`javax.wireless.messaging.cbs.receive`	Receive a Cell Broadcast Service (CBS) message
`javax.wireless.messaging.sms.receive`	Receive an SMS message
`javax.wireless.messaging.sms.send`	Send an SMS message
`jdk.dio.adc.ADCPermission`	Use of analog-to-digital converter (ADC)
`jdk.dio.atcmd.ATPermission`	Use of AT communication line
`jdk.dio.counter.CounterPermission`	Use of the hardware counter
`jdk.dio.dac.DACPermission`	Use of digital-to-analog converter (DAC)
`jdk.dio.DeviceMgmtPermission`	Opening of any Device I/O peripheral.
`jdk.dio.generic.GenericPermission`	Use of generic Device I/O connections
`jdk.dio.gpio.GPIOPinPermission`	Use of a General Purpose I/O (GPIO) pin
`jdk.dio.gpio.GPIOPortPermission`	Use of a General Purpose I/O (GPIO) port
`jdk.dio.i2cbus.I2CPermission`	Use of the I2C bus on the board
`jdk.dio.mmio.MMIOPermission`	Use of the Memory-Mapped I/O (MMIO) capabilities on the board
`jdk.dio.pwm.PWMPermission`	Use of the Pulse Width Modulation (PWM) capabilities on the board
`jdk.dio.spibus.SPIPermission`	Use of the SPI bus on the board
`jdk.dio.uart.UARTPermission`	Use of the UART bus on the board
`jdk.dio.watchdog.WatchdogTimerPermission`	Use of the watchdog timer on the board

Accessing Peripherals

Applications that require access to Device I/O APIs must request appropriate permissions in JAD files. For more information on using the Device I/O APIs, please see the *Device I/O API 1.1* specification and the associated Javadocs at the following site:

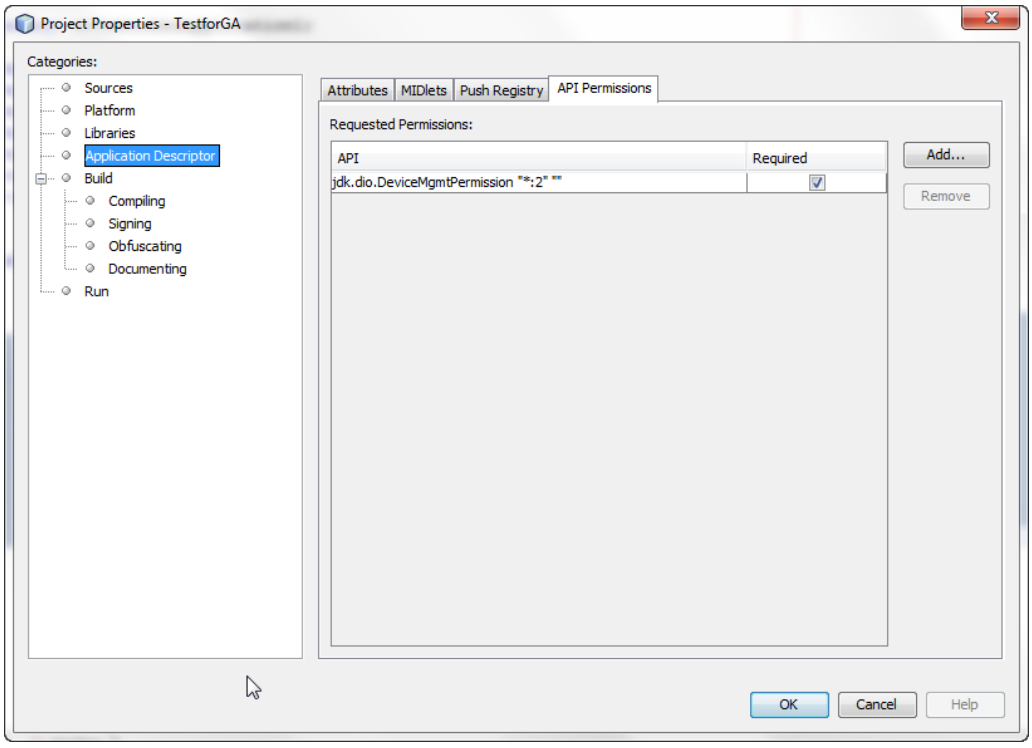
`http://docs.oracle.com/javame/`

Signing the Application with API Permissions

First, the JAD file must have the proper API permissions. Here is how to sign the application both in NetBeans and without an IDE.

- In **NetBeans**, right-click the project name and choose **Properties**. Select **Application Descriptor**, then in the resulting pane, select **API Permissions**. Click the **Add...** button, and add the appropriate permissions, as shown in Figure 3-1. Click **OK** to close the project properties dialog.

Figure 3-1 Adding Permissions Using the NetBeans IDE



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

```
MIDlet-Permission-1: com.oracle.dio.DeviceMgmtPermission "*:*" "open"
```

Method #1: Signing Application Using the NetBeans IDE

The NetBeans IDE enables developers both to sign the applications with a local certificate and upload the certificate on the device. See the appropriate *Getting Started Guide* for your embedded platform to learn how to use the NetBeans IDE to sign your application.

Method #2: Signing Application Using a Command Line

Signing applications using a command line is the preferred route for applications that are widely distributed. 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 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. Copy the `certs` directory from the board over to the desktop using an `sftp` client or `scp` command, change into the `certs` directory, 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 8.3 distribution.

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

   This command imports the information in `mykeystore.ks` that you just created to the `_main.ks` keystore. After this is completed, copy the `certs` directory back to the board by using an `sftp` client or `scp` command.

3. Use the following commands to sign your application before deploying it to the board:

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

   To sign with the SHA256 signature algorithm, add the `-useSha256` parameter. If not present, the default algorithm SHA1withRSA is used.

   ```
   jadtool -addjarsig -chainnum 1 -jarfile myjar.jar -alias myalias -
   keystore mykeystore.ks -storepass spass -keypass kpass -inputjad
   myjad.jad -outputjad myjad.jad -useSha256
   ```

Method #3: Using NullAuthenticationProvider

This method allows to bypass a certificate check and execute unsigned applications as if they were signed and given all requested permissions. This method should be used only for development and debugging. Final testing must be done using a real certificate as described in method #1.

To use `NullAuthenticationProvider`, set the following property in the `jwc_properties.ini` file on the board:

```
[internal]
authentication.provider = com.oracle.meep.security.NullAuthenticationProvider
```

Note that the Java runtime must not be running when editing the `jwc_properties.ini` file.

CLDC Permissions

The following permissions are available that affect the use of portions of the CLDC libraries.

FilePermission

The `java.io.FilePermission` controls access to a file or directory. A `FilePermission` consists of a pathname and a set of actions that are valid for the resource specified by that pathname.

Resource Name

The resource name is simply the pathname of the file or directory granted the specified actions. A pathname that ends in "`/*`" (where "`/`" is the file separator character, `File.separatorChar`) indicates all the files and directories contained in that directory. A pathname that ends with "`/-`" indicates all files and all recursive subdirectories contained in that directory. A pathname consisting of the special token "`<<ALL FILES>>`" matches any file.

Note:

A pathname need not have a leading "`/`". A pathname consisting of a single "`*`" indicates all the files in the current directory, while a pathname consisting of a single "`-`" indicates all the files in the current directory and recursively all files and subdirectories contained in the current directory.

Actions

Table 3-5 shows the actions can be requested with this permission, as a list of comma-separated keywords:

Table 3-2 FilePermission Actions

Value	Meaning
`read`	Read permission
`write`	Write permission
`execute`	Execute permission
`delete`	Permission to delete the resource
`readlink`	Read a link permission. This is retained for SE compatibility but is not currently used.

RuntimePermission

The `java.lang.RuntimePermission` represents runtime permissions. A `RuntimePermission` contains a resource name, but no actions list.

Resource Name

The resource name is the name of the runtime permission. The naming convention follows the hierarchical property naming convention. Also, an asterisk may appear at

the end of the name, following a "`.`", or by itself, to signify a wildcard match. For example: "`loadLibrary.*`" and "`*`" signify a wildcard match, while "`*loadLibrary`" and "`a*b`" do not.

Table 3-3 shows the possible runtime permissions that are allowed, as well as their effects and possible risks of using them.

Table 3-3 RuntimePermission Actions

Value	Effect	Risks
`exitVM.{exit status}`	Halting of the Java Virtual Machine (JVM) with the specified exit status	This allows an attacker to mount a denial-of-service attack by automatically forcing the virtual machine to halt. Note that the "`exitVM.*`" permission is automatically granted to all code loaded from the application class path, thus enabling applications to terminate themselves. Also, the "`exitVM`" permission is equivalent to "`exitVM.*`".
`setSecurityManager`	Setting of the security manager (possibly replacing an existing security manager)	The security manager is a class that allows applications to implement a security policy. Granting the `setSecurityManager` permission would allow code to change which security manager is used by installing a different, possibly less restrictive security manager, thereby bypassing checks that would have been enforced by the original security manager.
`createSecurityManager`	Creation of a new security manager	This gives code access to protected, sensitive methods that may disclose information about other classes or the execution stack.
`setIO`	Setting of `System.out` and `System.err`	This allows changing the value of the standard system streams. An attacker may set `System.err` to a null `OutputStream`, which would hide any error messages sent to `System.err`.
`modifyThread`	Modification of threads, possibly via calls to perform thread interrupts, or `setPriority()` and `setName()` methods	This allows an attacker to modify the behavior of any thread in the system.

LoggingPermission

The `java.util.logging.LoggingPermission` is a permission which the security manager will check when code that is running with a security manager calls one of the logging control methods, such as `Logger.setLevel()`.

Currently there is only one over-arching `LoggingPermission`, without resources or actions. This permission simply grants the ability to control the logging configuration, for example by adding or removing handlers, by adding or removing filters, or by changing logging levels.

PropertyPermission

The `java.util.PropertyPermission` is for general Java property permissions.

Resource Name

The resource name is the name of the property (for example, "`java.home`" or "`os.name`"). The naming convention follows the hierarchical property naming convention. Also, an asterisk may appear at the end of the name, following a "`.`", or by itself, to signify a wildcard match. For example: "`java.*`" and "`*`" signify a wildcard match, while "`*java`" and "`a*b`" do not.

Actions

Table 3-4 shows the actions can be requested with this permission, as a list of comma-separated keywords:

Table 3-4 PropertyPermission Actions

Value	Meaning
`read`	Read permission
`write`	Write permission

Care should be taken before granting code permission to access certain system properties. For example, granting permission to access the "`java.home`" system property gives potentially malevolent code sensitive information about the system environment, such as the Java installation directory. Also, granting permission to access the "`user.name`" and "`user.home`" system properties gives potentially malevolent code sensitive information about the user environment, including the user's account name and home directory.

Keystore Permissions

The following permissions are available that allow access to the Java ME keystore.

KeyStorePermission

The `com.oracle.crypto.keystore.KeyStorePermission` controls the type of access allowed to the key store.

Resource Name

Table 3-5 shows the resource names that can be requested with this permission:

Table 3-5 KeyStorePermission Resource Names

Value	Meaning
`client_only`	Access to client certificates only
*	Access to the entire certificate storage.

Device I/O Permissions

The following are among the more common permissions that can be requested from most Oracle Java ME Embedded devices, depending on whether the functionality is supported by the underlying board. See the Getting Started Guide for your embedded board to determine which Device I/O permissions and resources are available for use.

ADCPermission

The `jdk.dio.adc.ADCPermission` class defines permissions for Analog-to-Digital channel access on an embedded board.

Resource Name

The resource name is a numerical channel number. Refer to the Getting Started Guide of your embedded board to determine which channel numbers are available for ADC control.

Actions

Table 3-6 shows the actions can be requested with this permission:

Table 3-6 ADCPermission Actions

Value	Meaning
`open`	The requested channel is opened and available for use.
`powermanage`	Manage the power saving mode of a device.

ATPermission

The `jdk.dio.atcmd.ATPermission` class defines permissions AT device access.

Resource Name

The resource name is a numerical channel number. Refer to the Getting Started Guide of your embedded board to determine which channels are available for AT control.

Actions

Table 3-7 shows the actions can be requested with an `ATPermission`:

Table 3-7 ATPermission Actions

Value	Meaning
`open`	Open AT functions

Table 3-7 (Cont.) ATPermission Actions

Value	Meaning
`data`	Open data connections
`powermanage`	Manage the power saving mode of a device.

CounterPermission

The `jdk.dio.counter.CounterPermission` class defines permissions for pulse counter access.

Resource Name

The resource name is a numerical channel number. Refer to the Getting Started Guide of your embedded board to determine which channels are available for pulse counter control.

Actions

Table 3-8 shows the actions can be requested with an `ATPermission`:

Table 3-8 CounterPermission Actions

Value	Meaning
`open`	Open and access pulse counter functions
`powermanage`	Manage the power saving mode of a device.

DACPermission

The `jdk.dio.dac.DACPermission` class defines permissions for Digital-to-Analog channel access on an embedded board.

Resource Name

The resource name is a numerical channel number. Refer to the Getting Started Guide of your embedded board to determine which channel numbers are available for DAC control.

Actions

Table 3-9 shows the actions can be requested with this permission:

Table 3-9 DACPermission Actions

Value	Meaning
`open`	The requested channel is opened and available for use.
`powermanage`	Manage the power saving mode of a device.

DeviceMgmtPermission

The `jdk.dio.DeviceMgmtPermission` class defines permissions for registering and un-registering devices as well as opening devices using their registered configurations.

Resource Name

The resource name is a combination of a device name and of a device ID or range of device IDs. It takes the following form:

{device-name-spec} [":"*{device-id-spec}*]

{device-name-spec}

The *{device-name-spec}* string takes the following form:

{device-name} | "*" | ""

The *{device-name}* string is a device name that is returned by a call to `DeviceDescriptor.getName()`.

A *{device-name-spec}* specification consisting of the asterisk ("*") matches all device names. A *{device-name-spec}* specification consisting of the empty string ("") designates an undefined device name that may only be matched by an empty string or an asterisk.

{device-id-spec}

The *{device-id-spec}* string takes the following form:

{device-id} | "-"*{device-id}* | *{device-id}*"-"[*{device-id}*] | "*"

The *{device-id}* string is a device ID that is returned by a call to `DeviceDescriptor.getID()`. Note that the characters in the string must all be decimal digits.

A *{device-id-spec}* specification of the form "*n*-" (where *n* is a device ID) signifies all device IDs numbered *n* and above, while a specification of the form "-*n*" indicates all device IDs numbered *n* and below. A single asterisk in the place of the *{device-id-spec}* field matches all device IDs.

The name "*:*" matches all device names and all device IDs, as is the name "*".

Actions

Table 3-10 shows the actions can be requested with this permission:

Table 3-10 DeviceMgmtPermission Actions

Value	Meaning
`open`	Open a device using its device name or ID
`register`	Register a new device.
`unregister`	Un-register a new device

GenericPermission

The `jdk.dio.generic.GenericPermission` class defines permissions for generic device access on an embedded board.

Resource Name

The resource name is a numerical channel number. Refer to the Getting Started Guide of your embedded board to determine which channel numbers are available for generic devices.

Actions

Table 3-11 shows the actions can be requested with this permission:

Table 3-11 GenericPermission Actions

Value	Meaning
`open`	The requested channel is opened and available for use.
`powermanage`	Manage the power saving mode of a generic device.

GPIOPinPermission

The `jdk.dio.gpio.GPIOPinPermission` class defines permissions for General Purpose I/O (GPIO) pin access on an embedded board.

Resource Name

The resource name is a numerical pin number. Refer to the Getting Started Guide of your embedded board to determine which pin numbers are available for GPIO control.

Actions

Table 3-12 shows the actions can be requested with this permission:

Table 3-12 GPIOPinPermission Actions

Value	Meaning
`open`	The requested channel is opened and available for use.
`setdirection`	Request permission to change the GPIO pin direction
`powermanage`	Manage the power saving mode of a GPIO pin.

GPIOPortPermission

The `jdk.dio.gpio.GPIOPortPermission` class defines permissions for General Purpose I/O (GPIO) port access on an embedded board. A GPIO port is made up of several (typically eight) GPIO pins.

Resource Name

The resource name is a numerical port number. Refer to the Getting Started Guide of your embedded board to determine which port numbers are available for GPIO control.

Actions

Table 3-13 shows the actions can be requested with this permission:

Table 3-13 GPIOPortPermission Actions

Value	Meaning
`open`	The requested channel is opened and available for use.
`setdirection`	Request permission to change the GPIO port direction
`powermanage`	Manage the power saving mode of a GPIO port.

I2CPermission

The `jdk.dio.i2cbus.I2CPermission` class defines permissions for I2C bus access on an embedded board.

Resource Name

The resource name is a channel number. Refer to the Getting Started Guide of your embedded board to determine which channel numbers are available for I2C control.

Actions

Table 3-14 shows the actions can be requested with this permission:

Table 3-14 I2CPermission Actions

Value	Meaning
`open`	The requested channel is opened and available for use.
`powermanage`	Manage the power saving mode of an I2C bus.

MMIOPermission

The `jdk.dio.mmio.MMIOPermission` class defines permissions for MMIO bus access on an embedded board.

Resource Name

The resource name is a memory-address (in hexadecimal format) returned by a call to `MMIODeviceConfig.getAddress()`. The characters in the string must all be hexadecimal digits. Refer to the Getting Started Guide of your embedded board to determine which addresses are available for MMIO use.

Actions

Table 3-15 shows the actions can be requested with this permission:

Table 3-15 MMIOPermission Actions

Value	Meaning
`open`	The requested channel is opened and available for use.
`powermanage`	Manage the power saving mode of an MMIO bus.

PWMPermission

The `jdk.dio.pwm.PWMPermission` class defines permissions for Pulse Width Modulation (PWM) channel access on an embedded board.

Resource Name

The resource name is a numerical channel number. Refer to the Getting Started Guide of your embedded board to determine which channel numbers are available for PWM control.

Actions

Table 3-16 shows the actions can be requested with this permission:

Table 3-16 PWMPermission Actions

Value	Meaning
`open`	The requested channel is opened and available for use.
`powermanage`	Manage the power saving mode of a device.

SPIPermission

The `jdk.dio.spibus.SPIPermission` class defines permissions for SPI bus access on an embedded board.

Resource Name

The resource name is a channel number. Refer to the Getting Started Guide of your embedded board to determine which channel numbers are available for SPI control.

Actions

Table 3-17 shows the actions can be requested with this permission:

Table 3-17 SPIPermission Actions

Value	Meaning
`open`	The requested channel is opened and available for use.
`powermanage`	Manage the power saving mode of an SPI bus.

UARTPermission

The `jdk.dio.uart.UARTPermission` class defines permissions for UART bus access on an embedded board.

Resource Name

The resource name is a channel number. Refer to the Getting Started Guide of your embedded board to determine which channel numbers are available for UART control.

Actions

Table 3-18 shows the actions can be requested with this permission:

Table 3-18 UARTPermission Actions

Value	Meaning
`open`	The requested channel is opened and available for use.
`powermanage`	Manage the power saving mode of an UART bus.

WatchdogTimerPermission

The `jdk.dio.watchdog.WatchdogTimerPermission` class defines permissions for the watchdog timer on an embedded board.

Resource Name

The resource name is a channel number. Refer to the Getting Started Guide of your embedded board to determine which channel number is available for the watchdog timer.

Actions

Table 3-19 shows the actions can be requested with this permission:

Table 3-19 WatchdogTimerPermission Actions

Value	Meaning
`open`	The requested channel is opened and available for use.
`powermanage`	Manage the power saving mode of the watchdog timer..

Smart Cards

The following permission allows access to smart cards on Java ME embedded devices.

APDUPermission

The `javax.microedition.apdu.APDUPermission` class represents access to a smart card using the APDU protocol. An `APDUPermission` contains a resource name (also called a target name) but no actions list. The target name is the symbolic name of the `APDUPermission`.

Resource Name

The resource name can be one of two items, as shown in Table 3-20.

Table 3-20 APDUPermission Target Names

Target Name	Permission Allows
`aid`	The ability to communicate with a smart card application identified by an AID target.

Table 3-20 (Cont.) APDUPermission Target Names

Target Name	Permission Allows
`sat`	The ability to communicate with a (U)SAT application on channel 0.

Cellular

The following permissions deal with embedded devices that can connect to a cellular network.

CellularPermission

The `javax.microedition.cellular.CellularPermission` class defines permissions for cellular network resources on an embedded board. It consists only of a resource name.

Resource Name

The resource name can be one of three items, as shown in Table 3-21.

Table 3-21 CellularPermission Resource Names

Resource	Meaning
`subscriber`	Resources that access or modify the cellular subscriber identity, which is often recorded on a SIM, R-UIM, or CSIM.
`network`	Resources that access the cellular network.
*	All available cellular resources.

Generic Events

The following permissions deal with generic events that can be sent from the underlying runtime operating system to the Oracle Java ME Embedded runtime.

EventPermission

The `javax.microedition.event.EventPermission` class defines permissions that allow applications to receive events from the underlying runtime operating system.

Resource Name

The resource name is the name of the event, such as "`BATTERY_LEVEL`" or "`com.MyCompany.MyEvent`". The naming convention follows a hierarchical property naming convention. Also, an asterisk may appear at the *end* of the name, following a "`.`", or by itself, to signify a wildcard match. For example, "`com.MyCompany.*`" or "`*`" is valid, while "`*MyCompany`" or "`a*b`" is not valid.

Actions

The actions to be granted are a list of comma-separated keywords. The possible keywords are "`post`", "`postsystem`", "`read`" and "`register`". Table 3-22 gives more details on these keywords.

Table 3-22 EventPermission Actions

Value	Meaning
`post`	Permission to post an event.
`postsystem`	Permission to post a system event. To see which system events are supported, call `EventManager.getSystemEventNames()`.
`read`	Permission to read an event.
`readregister`	Permission to register and un-register applications to launch in response to events.

COMM Protocol

The following permissions deal with embedded devices that can use a COMM protocol through a serial port.

CommProtocolPermission

The `javax.microedition.io.CommProtocolPermission` class defines permissions for COMM resources on an embedded board. It consists only of a resource name.

Resource Name

The resource name is a base connection string and is typically formatted as:

`comm:`<*port identifier*>[<*optional parameters*>]

An exact BNF grammar for the COMM protocol URI is given in Table 3-23.

Table 3-23 CellularPermission Resource Names

Resource	Meaning
base connection string	"`comm:`"<*port_id*>[<*options_list*>] | "`comm:`"<*wildcarded_port_id*>
<*port_id*>	A non-empty case-sensitive string of alphanumeric characters
<*wildcarded_port_id*>	All available cellular resources.
<*options_list*>	*(<*baud_rate_string*>| <*bitsperchar*>| <*stopbits*>| <*parity*>| <*blocking*>| <*autocts*>| <*autorts*>)
<*baud_rate_string*>	"`;baudrate=`"<*baud_rate*>
<*baud_rate*>	non-empty string of digits
<*bitsperchar*>	"`;bitsperchar=`"<*bit_value*>

Table 3-23 (Cont.) CellularPermission Resource Names

Resource	Meaning
<*bit_value*>	"7" | "8"
<*stopbits*>	";stopbits="<*stop_value*>
<*stop_value*>	"1" | "2"
<*parity*>	";parity="<*parity_value*>
<*parity_value*>	"even" | "odd" | "none"
<*blocking*>	";blocking="<*on_off*>
<*autocts*>	";autocts="<*on_off*>
<*autorts*>	";autorts="<*on_off*>
<*on_off*>	"on" | "off"

Connector

The following permissions deal with those associated with the `javax.microedition.io.Connector` class, a factory class for creating new Connection objects.

CBS

The `javax.microedition.io.Connector.cbs` defines permissions for cellular broadcast service.

Resource Name

The resource name is a channel number. Refer to the Getting Started Guide of your embedded board to determine which channel number is available for the CBS.

Actions

Table 3-24 shows the actions can be requested with this permission:

Table 3-24 Connector CBS Actions

Value	Meaning
`open`	The requested channel is opened and available for use.
`powermanage`	Manage the power saving mode of the CBS..

File Read

The `javax.microedition.io.Connector.file.read` defines permissions for connections that read files.

Resource Name

The resource name is a channel number. Refer to the Getting Started Guide of your embedded board to determine which channel number is available for reading files.

Actions

Table 3-25 shows the actions can be requested with this permission:

Table 3-25 Connector File Read Actions

Value	Meaning
`open`	The requested channel is opened and available for use.
`powermanage`	Manage the power saving mode of the file read..

File Write

The `javax.microedition.io.Connector.file.write` defines permissions for connections that write files.

Resource Name

The resource name is a channel number. Refer to the Getting Started Guide of your embedded board to determine which channel number is available.

Actions

Table 3-26 shows the actions can be requested with this permission:

Table 3-26 WatchdogTimerPermission Actions

Value	Meaning
`open`	The requested channel is opened and available for use.
`powermanage`	Manage the power saving mode of the file write..

RTSP

The `javax.microedition.io.Connector.rtsp` defines permissions for connections that use the real-time streaming protocol (RTSP).

Resource Name

The resource name is a channel number. Refer to the Getting Started Guide of your embedded board to determine which channel number is available.

Actions

Table 3-27 shows the actions can be requested with this permission:

Table 3-27 WatchdogTimerPermission Actions

Value	Meaning
`open`	The requested channel is opened and available for use.

Table 3-27 (Cont.) WatchdogTimerPermission Actions

Value	Meaning
`powermanage`	Manage the power saving mode of the RTSP..

SMS

The `javax.microedition.io.Connector.sms` defines permissions for SMS messaging.

Resource Name

The resource name is a channel number. Refer to the Getting Started Guide of your embedded board to determine which channel number is available for SMS.

Actions

Table 3-28 shows the actions can be requested with this permission:

Table 3-28 WatchdogTimerPermission Actions

Value	Meaning
`open`	The requested channel is opened and available for use.
`powermanage`	Manage the power saving mode of the SMS..

Datagram Protocol

The following permissions deal with embedded devices that can use datagram protocols.

DatagramProtocolPermission

The `javax.microedition.io.DatagramProtocolPermission` class represents access rights to connections via the Datagram protocol. A `DatagramProtocolPermission` consists of a URI string, but no actions.

The URI string specifies a connection for sending and receiving datagrams. It takes the following general form:

`datagram://`*{host}*`:`*{portspec}* | `datagram://[:`*{portspec}*`]`

The value of the *{host}* field must be a symbolic hostname, a literal IPv4 address or an IP-literal as specified by RFC 3986. An IP-literal requires an IPv6Address to bew surrounded with square brackets (`[]`). Note that IPvFuture addresses from RFC 3986 are not currently supported.

The *{host}* field is omitted to indicate an inbound, server-mode connection. Server-mode URIs may also omit the *{portspec}* field to request a system-assigned port number. In such a case, the `DatagramProtocolPermission` is normalized to the equivalent URI: `datagram://:1024-65535`.

If the *{host}* string is a DNS name, an asterisk may appear in the left-most position to indicate a match of 1 or more entire domain labels. Partial domain label matches are not permitted. For example, "`*.oracle.com`" is valid, but "`*oracle.com`" is not. An asterisk by itself matches all hosts in outbound, client-mode connections.

The *{portspec}* string takes the following form:

portnumber | `-`*portnumber* | *portnumber*`-`[*portnumber*] | "`*`"

A *{portspec}* of the form "*n*`-`" (where *n* is a port number) signifies all ports numbered *n* and above, while a specification of the form "`-`*n*" indicates all ports numbered *n* and below. A single asterisk in the place of the *{portspec}* field matches all ports. Therefore, the URI "`datagram://:*`" matches server-mode datagram connections to all ports, and the URI "`datagram://*:*`" matches client-mode datagram connections to all hosts on all ports.

DTLSProtocolPermission

The `javax.microedition.io.DTLSProtocolPermission` class represents access rights to connections that use the Datagram Transport Layer Security (DTLS) protocol. A `DTLSProtocolPermission` consists of a URI string but no actions list.The URI string specifies a connection for sending and receiving datagrams. It takes the following general form:

`dtls://`*{host}*`:`*{portspec}* The value of the *{host}* field must be a symbolic hostname, a literal IPv4 address or an IP-literal as specified by RFC 3986. An IP-literal requires an IPv6Address to be surrounded with square brackets (`[]`). Note that IPvFuture addresses from RFC 3986 are not supported.

If the *{host}* string is a DNS name, an asterisk may appear in the left-most position to indicate a match of 1 or more entire domain labels. Partial domain label matches are not permitted. For example, "`*.oracle.com`" is valid, but "`*oracle.com`" is not. An asterisk by itself matches all hosts in outbound, client-mode connections.

The *{portspec}* string takes the following form:

portnumber | `-`*portnumber* | *portnumber*`-`[*portnumber*] | "`*`"

A *{portspec}* of the form "*n*`-`" (where *n* is a port number) signifies all ports numbered *n* and above, while a specification of the form "`-`*n*" indicates all ports numbered *n* and below. A single asterisk in the place of the *{portspec}* field matches all ports. Therefore, the URI "`dtls://*:*`" matches client-mode datagram connections to all hosts on all ports.

DTLSServerPermission

The `javax.microedition.io.DTLSServerPermission` class represents access rights to server connections via the the Datagram Transport Layer Security (DTLS) protocol protocol. A `DTLSServerPermission` consists of a URI string but no actions list. The URI string specifies a connection for sending and receiving datagrams. It takes the following general form:

`dtls://:`*{portspec}*

The exact syntax for the `DTLSSrverPermission` URI is provided by this BNF.

The *{portspec}* string takes the following form:

portnumber | `-`*portnumber* | *portnumber*`-`[*portnumber*] | "`*`"

A *{portspec}* of the form "*N*`-`" (where *N* is a port number) signifies all ports numbered *N* and above, while a specification of the form "`-`*N*" indicates all ports numbered *N* and below. A single asterisk in the place of the *{portspec}* field matches all ports. Therefore, the URI "`dtls://*`" matches secure-mode secure datagram connections to all ports.

File Protocol

The following permissions deal with embedded devices that can use files.

FileProtocolPermission

The `javax.microedition.io.FileProtocolPermission` class represents access rights to connections via the "file" protocol. A `FileProtocolPermission` consists of a URI string indicating a fully-qualified, absolute pathname as well as a set of actions desired for that pathname.

Resource Name

The URI string takes the following general form:

`file://`[*{host}*]*{absolute_path}* | `file:`*{absolute_path}*

The exact syntax is given by RFCs 1738 and 2396. In addition, a pathname that ends in "`/*`" matches all the files and directories contained in that directory. A pathname that ends with "`/-`" recursively matches all files and subdirectories contained in that directory.

In addition to the syntax defined by RFC 1738, `FileProtocolPermission` must accept and normalize URIs of the form `file:`*{abs_path}*. If *{host}* is omitted, it is equivalent to using `localhost`. Also, note that *{absolute_path}* follows the syntax defined for *{fpath}* in RFC 1738.

Actions

Table 3-29 shows the actions can be requested with this permission. Note that multiple actions can be requested by separating keywords with commas.

Table 3-29 FileProtocolPermission Actions

Value	Meaning
`read`	The file can be read from using the protocol.
`write`	The file can be written to using the protocol.

Hypertext Transfer Protocols

The following permissions deal with embedded devices that can use HTTP or HTTPS protocols.

HTTPProtocolPermission

The `javax.microedition.io.HTTPProtocolPermission` class represents access rights to connections via the HTTP protocol. An `HttpProtocolPermission` consists of a URI string, but no actions list.

The URI string specifies a data resource accessible via HTTP. It takes the following general form:

`http://`*{host}*[`:`*{portspec}*][*{pathname}*][`?`*{query}*][`#`*{fragment}*]

The value of the *{host}* field must be a symbolic hostname, a literal IPv4 address or an IP-literal as specified by RFC 3986. An IP-literal requires IPv6Address to be

surrounded with square brackets ([]). IPvFuture addresses from RFC 3986 are not supported.

If the *{host}* string is a DNS name, an asterisk may appear in the left-most position to indicate a match of one or more entire domain labels. Partial domain label matches are not permitted. For example, "`*.oracle.com`" is valid, but "`*oracle.com`" is not. An asterisk by itself matches all hosts.

The *{portspec}* string takes the following form:

portnumber | *-portnumber* | *portnumber*`-`[*portnumber*] | `*` | *empty string*

A *{portspec}* specification of the form "*n*-" (where *n* is a port number) signifies all ports numbered *n* and above, while a specification of the form "-*n*" indicates all ports numbered *n* and below. A single asterisk in the place of the *{portspec}* field matches all ports; therefore, the URI "`http://*:*`" matches HTTP connections to all hosts on all ports. If the *{portspec}* field is omitted, default port 80 is assumed.

HTTPSProtocolPermission

The `javax.microedition.io.HTTPSProtocolPermission` class represents access rights to connections via the HTTPS protocol. A `HttpsProtocolPermission` consists of a URI string, but no actions list.

The URI string specifies a data resource accessible via secure HTTPS. It takes the following general form:

`http://`*{host}*[`:`*{portspec}*][*{pathname}*][`?`*{query}*][*#{fragment}*]

The value of the *{host}* field must be a symbolic hostname, a literal IPv4 address or an IP-literal as specified by RFC 3986. An IP-literal requires IPv6Address to be surrounded with square brackets ([]). IPvFuture addresses from RFC 3986 are not supported.

If the *{host}* string is a DNS name, an asterisk may appear in the left-most position to indicate a match of one or more entire domain labels. Partial domain label matches are not permitted. For example, "`*.oracle.com`" is valid, but "`*oracle.com`" is not. An asterisk by itself matches all hosts.

The *{portspec}* string takes the following form:

portnumber | *-portnumber* | *portnumber*`-`[*portnumber*] | `*` | *empty string*

A *{portspec}* specification of the form "*n*-" (where *n* is a port number) signifies all ports numbered *n* and above, while a specification of the form "-*n*" indicates all ports numbered *n* and below. A single asterisk in the place of the *{portspec}* field matches all ports; therefore, the URI "`https://*:*`" matches HTTPS connections to all hosts on all ports. If the *{portspec}* field is omitted, default port 443 is assumed.

IMC

The following permissions deal with embedded devices that use the Inter-MIDlet Communication (IMC) protocol.

IMCProtocolPermission

The `javax.microedition.io.IMCProtocolPermission` class defines permissions for inter-MIDlet communication on an embedded board. IMC uses a low-level asynchronous bi-directional stream connection for communication between applications. The permission consists only of a resource name.

Resource Name

The resource name consists of a number of rules to create a base client connection string; these rules are shown in Table 3-30.

Table 3-30 IMCProtocolPermission Resource Name Rules

Rule	Meaning
Base client connection string	"`imc://`" (<*Application UID*> | "`*`") "`:`" <*server name*> "`:`" <*server version*> "`;`"
<*Application UID*>	<*Application suite vendor*>"`:`" <*Application suite name*>"`:`" <*Application suite version*>
<*Application suite vendor*>	:The application suite vendor
<*Application suite name*>	The application suite name
<*Application suite version*>	Formatted application suite version or wildcard character "`*`"
<*server name*>	IMC server name following the class naming syntax
<*server version*>	The version of the IMC server. Version backward compatibility is assumed.Versioning follows the format defined for the `MIDlet-Version` attribute.

Note that in the first rule, the wildcard "`*`" may be used instead of a specific <*Application UID*> when opening an IMC client connection. When the wildcard character is used, it allows the client to connect to any applications (even those from different vendors) if they all provide the same IMC service and meet the authorization requirements. However, which application's IMC server the client will be connected to is implementation specific.

Multicast Protocols

The following permissions deal with embedded devices that use the multicast protocols.

MulticastProtocolPermission

The `javax.microedition.io.MulticastProtocolPermission` class represents access rights to connections via the "multicast" protocol. A `MulticastProtocolPermission` consists of a URI string but no actions list.

The exact syntax for the `MulticastProtocolPermission` URI is provided by rules shown in Table 3-31.

Table 3-31 MulticastProtocolPermission Resource Name Rules

Rule	Meaning
base multicast connection string	<*inbound_connection*> | <*outbound_connection*>
<*inbound_connection*>	"`multicast://:`" [<*portnumber*>] [<*auto_join*>]
<*outbound_connection*>	"`multicast://`" <*host*> "`:`" <*portnumber*>

Table 3-31 (Cont.) MulticastProtocolPermission Resource Name Rules

Rule	Meaning
<*multicast_permission*>	"`multicast://`"[<*hostspec*>]":" <*portspec*>
<*host*>	<*host name*> | <*ipaddr*>
<*ipaddr*>	*IPv4 address* | '[' *IPv6 address* ']'
<*hostspec*>	<*host*> | "*" .
<*auto_join*>	"`?join=`"<*host*>
<*portspec*>	<*portnumber*> | <*portrange*> | "*"
<*portnumber*>	*numeric port number*
<*portrange*>	<*portnumber*> "-" | "-" <*portnumber*> | <*portnumber*> "-" <*portnumber*>

The value of the *{host}* field must be a symbolic hostname, a literal IPv4 multicast address or a literal IPv6 address surrounded by square brackets (`[]`), as specified by RFC 3986. The *{hostspec}* may be "*" to allow connection to any multicast host group. The *{hostspec}* field may also be omitted to indicate an inbound, server-mode connection.

Server-mode URIs may also omit the <*portspec*> field to request a system-assigned port number. In such a case, the `MulticastProtocolPermission` is normalized to the equivalent URI "`multicast://:1024-65535`".

The <*portspec*> string takes the following form:

portnumber | -*portnumber* | *portnumber*-[*portnumber*] | "*"

A <*portspec*> specification of the form "*n*-" (where n is a port number) signifies all ports numbered *n* and above, while a specification of the form "-*n*" indicates all ports numbered *n* and below. A single asterisk in the place of the <*portspec*> field matches all ports. Therefore, the URI "`multicast://`<*ipaddr*>`:`" matches multicast a host group to all ports, and the URI "`multicast://*:*`" matches multicast connections to all host groups on all ports.

Push Protocols

The following permissions deal with embedded devices that use push protocols.

PushRegistryPermission

The `javax.microedition.io.PushRegistryPermission` class is used to check the static and dynamic registration of push connections and for registration of an alarm. The permission covers static registration via application attributes, and dynamic registration via `PushRegistry.registerConnection(...)` and alarm registration with `PushRegistry.registerAlarm()`.

For the purposes of Push Registration permission, the URI MUST consist only of the scheme and delimiter (":") as defined by RFC-3986. The scheme may contain the wildcard character "*", which allows registration of all schemes. For alarm registration, the URI is "*" and the action is alarm. Push registration and alarm

registration can be combined in a single permission. For example, the resource is "`file:`" and the actions are "`static,dynamic,alarm`".

Actions

Table 3-32 shows the actions can be requested with this permission. Note that multiple actions can be requested by separating keywords with commas.

Table 3-32 PushRegistryPermission Actions

Value	Meaning
`static`	Allows registration of a Push Connection in the packaging of the application suite
`dynamic`	Allows registration of a Push Connection using PushRegistry.registerConnection
`alarm`	Allows registration of an alarm using PushRegistry.registerAlarm

Socket Protocols

The following permissions deal with embedded devices that can use HTTP or HTTPS protocols.

SocketProtocolPermission

The `javax.microedition.io.SocketProtocolPermission` class represents access rights to connections via the "socket" protocol. A `SocketProtocolPermission` consists of a URI string but no actions list.

The URI string specifies a socket stream connection. It takes the following general form:

`socket://`*{host}*`:`*{portspec}* | `socket://[:`*{portspec}*`]`

The exact syntax for the `SocketProtocolPermission` URI is given by the grammar in Table 3-33.

Table 3-33 SocketProtocolPermission Resource Name Rules

Rule	Meaning
base socket connection string	"`socket://`"*<inbound_connection>* | "`socket://`"*<outbound_connection>*
<inbound_connection>	"`:` " | "`:`"`[` *<porspec>* `]` | *empty string*
<outbound_connection>	*<host>* "`:`" *<portspec>*
<host>	*<host name>* | *<ipaddr>* | *<wildcarded DNS>*
<ipaddr>	*IPv4 address* | '`[`' *IPv6 address* '`]`'
<wildcarded_DNS>	"`*`" *("`.`"*<domainlabel>*) | "`*`" *followed by zero or more internet domain labels, separated by* "`.`"
<domainlabel>	*internet domain label*

Table 3-33 (Cont.) SocketProtocolPermission Resource Name Rules

Rule	Meaning
<portspec>	*<portnumber>* | *<portrange>* | "*"
<portnumber>	*numeric port number*
<portrange>	*<portnumber>* "-" | "-" *<portnumber>* | *<portnumber>* "-" *<portnumber>*

The value of the *{host}* field must be a symbolic hostname, a literal IPv4 address or an IP-literal with an IPv6Address as specified by RFC 3986. An IPv6Address must be surrounded with square brackets (`[]`). Note that IPvFuture addresses are not currently supported.

The *{host}* field may be omitted to indicate a server-mode connection. Server-mode URIs may also omit the *{portspec}* field to indicate a system-assigned port number. In such a case, the `SocketProtocolPermission` is normalized to the equivalent URI "`socket://:1024-65535`".

If the *{host}* string is a DNS name, an asterisk may appear in the left-most position to indicate a match of one or more entire domain labels. Partial domain label matches are not permitted, therefore "`*.oracle.com`" is valid, but "`*oracle.com`" is not. An asterisk by itself matches all hosts in client-mode connections;

The *{portspec}* string takes the following form:

portnumber | "-" *portnumber* | *portnumber* "-" [*portnumber*] | "*"

A *{portspec}* specification of the form "*n*-" (where *n* is a port number) signifies all ports numbered *n* and above, while a specification of the form "-*n*" indicates all ports numbered *n* and below. A single asterisk may be used in place of the *{portspec}* field to indicate all ports. Therefore, the URI "`socket://:*`" matches server-mode socket connections to all ports, and the URI "`socket://*:*`" matches client-mode socket connections to all hosts on all ports.

Note:

The syntax of URLs accepted by `Connector.open()` for sockets differs from the syntax for `SocketProtocolPermission`. In the `socket:` protocol, the ":" delimiter must always be present even if there is no port number, whereas the delimiter must not be present unless there is a port number in `SocketProtocolPermission`.

SSLProtocolPermission

The `javax.microedition.io.SSLProtocolPermission` class represents access rights to connections that use the Secure Sockets Layer (SSL) protocol. A `SSLProtocolPermission` consists of a URI string but no actions list.

The URI string specifies a secure socket stream connection. It takes the following general form:

`ssl://`*{host}*`:`*{portspec}* | `ssl://`[*:{portspec}*]

The exact syntax for the `SSLProtocolPermission` URI is given in Table 3-34.

Table 3-34 SSLProtocolPermission Resource Name Rules

Rule	Meaning
base SSL connection string	"`ssl://`"<*inbound_connection*> | "`ssl://`"<*outbound_connection*>
<*inbound_connection*>	"`:` " | "`:`"[<*portspec*>] | *empty string*
<*outbound_connection*>	<*host*> "`:`" <*portspec*>
<*host*>	<*host name*> | <*ipaddr*> | <*wildcarded DNS*>
<*ipaddr*>	*IPv4 address* | '`[`' *IPv6 address* '`]`'
<*wildcarded_DNS*>	"`*`" *("`.`"<*domainlabel*>) | "`*`" *followed by zero or more internet domain labels, separated by* "`.`"
<*domainlabel*>	*internet domain label*
<*portspec*>	<*portnumber*> | <*portrange*> | "`*`"
<*portnumber*>	*numeric port number*
<*portrange*>	<*portnumber*> "`-`" | "`-`" <*portnumber*> | <*portnumber*> "`-`" <*portnumber*>

The value of the *{host}* field must be a symbolic hostname, a literal IPv4 address or an IP-literal as specified by RFC 3986. An IPv6Address must be surrounded with square brackets (`[]`). Note that IPvFuture addresses are not supported.

The *{host}* field is omitted to indicate a server-mode connection. Server-mode URIs may also omit the *{portspec}* field to indicate a system-assigned port number. In such a case, the `SSLProtocolPermission` is normalized to the equivalent URI "`ssl://:1024-65535`".

If the *{host}* string is a DNS name, an asterisk may appear in the left-most position to indicate a match of one or more entire domain labels. Partial domain label matches are not permitted, therefore "`*.oracle.com`" is valid, but "`*oracle.com`" is not. An asterisk by itself matches all hosts.

The *{portspec}* string takes the following form:

portnumber \| `-`*portnumber* \| *portnumber*`-`[*portnumber*] \| "`*`"

A *{portspec}* specification of the form "*n*`-`" (where n is a port number) signifies all ports numbered *n* and above, while a specification of the form "`-`*n*" indicates all ports numbered *n* and below. A single asterisk in the place of the *{portspec}* field matches all ports. Therefore, the URI "`ssl://:*`" matches secure server connections to all ports, and the URI "`ssl://*:*`" matches secure connections to all hosts on all ports.

Location

The following permissions allow location functionality on an embedded device.

LocationPermission

The `javax.microedition.LocationPermission` class is used to allow access to the location functionality of an embedded device. This permission consists of only the class, but no targets or actions.

Media

The following permissions deal with embedded devices that have the ability to record or playback media.

RecordControl

The `javax.microedition.media.RecordControl` class allows Java ME embedded applications to control audio recording on an embedded device. This permission consists of only the class, but no targets or actions.

VideoControl

The `javax.microedition.media.VideoControl.getSnapshot` permissions grants Java ME embedded applications the ability to take snapshot pictures on an embedded device. This permission consists of only the class, but no targets or actions.

Auto-Start

The following permissions allow auto-start functionality on an embedded device.

AutoStartPermission

The `javax.microedition.midlet.AutoStartPermission` allows applications in an application suite to assume the auto-start application behavior.

Resource Names

Table 3-35 shows the names that are allowed with this permission.

Table 3-35 AutoStartPermission Actions

Value	Meaning
`allowed`	Auto-start of the application is allowed
`not allowed`	Auto-start of the application is not allowed

Power

The following permission allows applications to access the power state functionality of an embedded device.

PowerStatePermission

The `javax.microedition.power.PowerStatePermission` allows calls to `PowerManager.setPowerState()` method.

Resource Names

Table 3-36 shows the names that are allowed with this permission.

Table 3-36 PowerStatePermission Actions

Value	Meaning
`set`	Calls to `setPowerState(..., false)` are allowed
`setUrgent`	Calls to `setPowerState(..., true)` are allowed

Software Management

The following permissions allow applications to use of the software management (SWM) APIs on an embedded device.

SWMPermission

The `javax.microedition.power.SWMPermission` provides permission handling for SWM API permissions. An `SWMPermission` object contains a resource and actions.

Resource Names

Table 3-37 shows the resource names that are allowed with this permission.

Table 3-37 SWMPermission Resource Names

Value	Meaning
`client`	Permission to perform the listed actions only for applications assigned to the same client
`crossClient`	Permission to perform the listed actions also for applications assigned to other clients. Usually this is a permission reserved for the root client. Granting this permissions to other clients should carefully considered to avoid security breaches.

Actions

The actions to be granted are a list of comma-separated keywords, as shown in Table 3-38, as well as whether they are permitted on a trusted and non-trusted client.

Table 3-38 SWMPermission Actions

Name and Action	Assigned to Trusted Client	Assigned to Non-Trusted Client
`client, manageSuite`	Permitted	Not Permitted.
`client, installation`	Permitted.	Not Permitted.
`client, manageTask`	Permitted.	Not Permitted.
`crossClient, manageSuite`	Permitted, but not recommended	Not Permitted.

Table 3-38 (Cont.) SWMPermission Actions

Name and Action	Assigned to Trusted Client	Assigned to Non-Trusted Client
`crossClient, installation`	Permitted, but not recommended	Not Permitted.
`crossClient, manageTask`	Permitted, but not recommended	Not Permitted.

Runtime Update

The following permission allows to update the Java ME runtime on an embedded device remotely.

RuntimeUpdatePermission

The `RuntimeUpdatePermission` allows calls to the `RuntimeUpdateManager.saveNewRuntime(java.io.InputStream)` and `RuntimeUpdateManager.scheduleUpdateAfterReboot()` methods.

Resource Names

shows the names that are allowed with this permission.

Table 3-39 RuntimeUpdatePermission Names

Value	Meaning
`save`	Calls to `saveNewRuntime` are allowed
`update`	Calls to `scheduleUpdateAfterReboot` are allowed

4

Software Management

This chapter introduces the Software Management (SWM) APIs of the Java ME Embedded Profile (MEEP) version 8. These APIs provided extended software management features for Oracle Java ME Embedded applications, as given in the `javax.microedition.swm` package. There are five interfaces and six classes in this package that can be used by applications to enhance software management. In addition, there are a number of enumerations that are present in the package; these are documented near the classes and methods that use them throughout this chapter.

SuiteInstallListener Interface

`SuiteInstallListener` is a sub-interface that provides progress data for an installer that is downloading an app or a link.

The interface consists of two methods, both of which are called at certain times during installation. One is the `installationDone()` method, which provides only a single code, the definitions of which can be found in the `InstallerErrorCode` interface. The other is the `updateStatus()` method, which identifies the current task as one of the `SuiteInstallStage` constants that are shown in Table 4-1, and provides an integer percentage of completeness.

Table 4-1 SuiteInstallState

Name	Description
`DONE`	Installation has completed
`DOWNLOADING_BODY`	Install stage: downloading application body.
`DOWNLOADING_DATA`	Install stage: downloading additional application data.
`DOWNLOADING_DESCRIPTOR`	Install stage: downloading application descriptor.
`STORING`	Install stage: storing application.
`VERIFYING`	Install stage: verifying downloaded content.

Here are the two method defined in the `SuiteInstallListener` interface:

- `void installationDone(int errorCode)`

  This method is called by the installer to report that the installation has completed. The resulting integer code is contained in the `InstallerErrorCode` class. See "InstallerErrorCode" for more information about installation error codes.

- `void updateStatus(SuiteManagementTracker tracker, SuiteInstallStage status, int percent)`

This method is called by the installer to inform the listener of the current status of the install. The stage is given by an integer constant as shown in Table 4-1. The percent is an integer between 0 and 100.

SuiteListener Interface

`SuiteListener` is an interface that provides a notification that the current state of a suite has changed.

There is only one method defined in the `SuiteListener` interface:

- `void notifySuiteStateChanged(SuiteManagementTracker tracker, SuiteState newState)`

  This method is called to notify a listener that the current state of a suite has changed. A reference to the current `SuiteManagementTracker` is included, as well as an instance of `SuiteState`, which indicates the new state.

SuiteManager Interface

The `SuiteManager` interface consists of only seven methods that add or remove suites, add or remove suite listeners, retrieve a list of the currently installed suites, or retrieve the current `SuiteInstaller`.

- `void addSuiteListener(SuiteListener theListener)`

  This method adds a `SuiteListener` object to the current `SuiteManager`.

- `Suite getSuite(java.lang.String vendor, java.lang.String name)`

  This method returns an instance of the currently installed `Suite`.

- `SuiteInstaller getSuiteInstaller(byte[] instData, int offset, int length, boolean ignoreUpdateLock)`

  This method returns the current SuiteInstaller.

- `SuiteInstaller getSuiteInstaller(java.lang.String locationUrl, boolean ignoreUpdateLock)`

  This method returns the current SuiteInstaller

- `java.util.List<Suite> getSuites(SuiteType type)`

  This method requests a list of installed suites of specified type.

- `void removeSuite(Suite suite, boolean ignoreRemoveLock)`

  This method removes a Suite.

- `void removeSuiteListener(SuiteListener theListener)`

  This method removes a SuiteListener.

TaskListener Interface

The `TaskListener` interface is an interface used to receive updates about a task that is currently running.

- `void notifyStatusUpdate(Task task, TaskStatus newStatus)`

This method is called when the current task has a new status update to report. The method passes a reference to the `Task` in question, as well as a `TaskStatus` object reporting the new status.

TaskManager Interface

The `TaskManager` interface is an interface used to manage the tasks.

- `void addTaskListener(TaskListener) throws SecurityException`

  This method adds a `TaskListener`.

- `Task getCurrentTask() throws SecurityException`

  This method returns the current task that is running.

- `java.util.List<Task> getTaskList(boolean includeSystem)`

  This method obtains a list of Task objects. If system tasks are to be included, that can be specified with the boolean parameter.

- `void removeTaskListener(TaskListener listener)`

  This method removes a TaskListener.

- `boolean setForegroundTask(Task task) throws java.lang.IllegalArgumentException`

  This method assigns the specified task to be the currently running foreground task. A task is said to be in the foreground if the LUI API or another UI API is supported and the task is visible on the display, or if the Key API is supported and input device events will be delivered to it. If none of those packages is supported by the implementation, a call to this method has no effect.

- `boolean setPriority(Task task, TaskPriority priority) throws java.lang.IllegalArgumentException`

  Changes the priority for the given task. The method returns true if the change was successful, or false otherwise.

- `Task startTask(Suite suite, String className) throws java.lang.IllegalArgumentException, java.lang.IllegalStateException`

  Starts a `Task` from the given class name in the given `Suite`. This method throws an exception if suite is a library and can therefore not be started. Calling this method schedules a new application execution. The new task is created with `TaskStatus.STARTING` on success or `TaskStatus.START_FAILED` on failure.

  More than one call to this method can be performed with the same arguments. In this case subsequent calls lead to attempts to re-start the task. In case of unsuccessful attempt to re-start the task, an appropriate exception is thrown or the corresponding state `TaskStatus.START_FAILED` is set to the returned task object.

- `boolean stopTask(Task task)` throws java.lang.IllegalArgumentException, java.lang.IllegalStateException

  This method cancels an installation that is in progress. It returns true if the cancellation was successful, or false otherwise.

ManagerFactory Class

The `ManagerFactory` class is a global factory that is used to obtain a `SuiteManager` or a `TaskManager` implementation.

- `static SuiteManager getSuiteManager()`

  This method returns an implementation of a `SuiteManager`.

- `static TaskManager getTaskManager()`

  This method returns an implementation of a `TaskManager`.

The Suite Class

All IMlet suites maintain a basic set of identification and state information that acts as a *descriptor*. This descriptor is represented by the `Suite` class.

Suites can be one of four types, presented in the `SuiteType` enumeration, and shown in Table 4-2:

Table 4-2 SuiteType Enumeration

Suite Type	Description
`ST_APPLICATION`	The suite contains one or more MIDlets with an entry point that can be executed.
`ST_LIBRARY`	The suite is a library that can be used by one or more applications.
`ST_SYSTEM`	The suite is a system-level application.
`ST_INVALID`	The suite is invalid and cannot be found or executed.

In addition, suites contain binary flags that describe their state, presented in the `SuiteStateFlag` enumeration, and shown in Table 4-3:

Table 4-3 SuiteStageFlag Enumeration

State	Description
`AVAILABLE`	The suite is available for use.
`ENABLED`	The suite is enabled. When a suite is disabled, any attempt to run application or use a library from this suite should fail.
`SYSTEM`	The suite is a system-level suite.
`PREINSTALLED`	The suite is a system-level suite that cannot be updated.
`REMOVE_DENIED`	The suite should not be removed.
`UPDATE_DENIED`	The suite should not be updated.

The following are method present in the `Suite` class.

- `java.lang.String getName()`

This method returns the name for the given suite.

- `java.lang.String getVendor()`

  This method returns the vendor for the given suite.

- `java.lang.String getVersion()`

  This method returns the version of the given suite.

- `java.lang.String getDownloadUrl()`

  This method returns the URL that the JAD or JAR was downloaded from.

- `java.util.Iteration<String> getAttributes()`

  This method returns a `String` array that provides the names of the available properties. The properties returned are those from the JAD file and the manifest combined into a single array.

- `java.lang.String getAttributeValue(String name)`

  This method retrieves the value for the respective attribute name.

- `SuiteType getSuiteType()`

  This method returns the suite type. See Table 4-2 for more information.

- `public boolean isSuiteState(SuiteStateFlag state)`

  This method checks the current state boolean to see if it is true.

- `public void setSuiteStateFlag(SuiteStateFlag state, boolean value) throws java.lang.IllegalArgumentException, java.lang.IllegalStateException, java.lang.SecurityException`

  This method sets the specified flag to the specified value. If a `Suite` has been created, `SuiteStateFlag.ENABLED` and `SuiteStateFlag.AVAILABLE` are always set to `true`, while `SuiteStateFlag.REMOVE_DENIED` and `SuiteStateFlag.UPDATE_DENIED` are set to `false`. These states can be changed by calling this method. The `SuiteStateFlag.SYSTEM` and `SuiteStateFlag.PREINSTALLED` flags are only set for system suites or pre-installed suites, respectively, and cannot be unset or set by this method. To be able to set suite flags, caller application should request `javax.microedition.swm.SWMPermission("client", "manageSuite")` or `javax.microedition.swm.SWMPermission ("crossClient", "manageSuite")` permission. See `SWMPermission` for more details.

- `public java.util.Iterator<java.lang.String> getMIDlets()`

  This method returns a list of the applications (application class names) in this suite. The first application in the enumeration is the default application as specified in the `MIDlet-1` field of the descriptor.

- `public java.util.Iterator<Suite> getDependencies()`

  This method returns a list of the shared libraries this `Suite` depends on

- `public boolean isTrusted()`

  Checks if this `Suite` is trusted or not. The return value is always true if it is a `SYSTEM_SUITE`.

- `public boolean isInstalled()`

Checks if this `Suite` is still installed or has been removed.

SuiteInstaller Class

The `ManagerFactory` class is a global factory that is used to obtain a `SuiteManager` or a `TaskManager` implementation.

- `void addInstallationListener(SuiteInstallListener listener)`

  This method adds a `SuiteInstallListener` to this suite installer

- `void removeInstallationListener(SuiteInstallationListener listener)`

  This method removes a `SuiteInstallListener` to this suite installer.

- `SuiteManagementTracker start()`

  This method starts installation of the suite. The installation can be the first installation of this suite, or a re-installation (update) of a suite that had been installed before. A `SuiteInstallListener` must be added in order to handle callback requests.This method returns an instance of `SuiteManagementTracker`; the caller can observe the progress of the installation via the `SuiteInstallListener` added. Please note that the method may not return quickly. Depending on the provisioning mechanism used in the implementation of MEEP 8, it may be necessary to download the entire JAR data first in order to inspect the manifest of the application suite in order to find out whether this is a new installation or an update of an existing application suite. Depending on the network connection, this may take some time.In case the previous attempt to install this suite (initiated by a previous call of the `start()` method) has not been finished at the time the new call takes place, the call is queued and the new attempt to install (in case the first one failed) or the re-installation (in case the first call was successful), respectively, starts as soon as the first installation attempt or installation has been finished.A new instance of `SuiteManagementTracker` will be created for every call to this method and assigned to the `Suite` to be installed as soon as the installation has been completed successfully. In case of an update of an existing `Suite`, the `SuiteManagementTracker` instance is assigned to the existing `Suite` object from the beginning.If the initiating application does not have the right `SWMPermission`, the installation will fail with `InstallErrorCodes.UNAUTHORIZED_INSTALL`.

- `void cancel()`

  Begins installation of the suite.

SuiteManagementTracker Class

An instance of this class is generated as soon as an installation or update of a `Suite` is started using `SuiteInstaller.start()`. Invoking that method creates a new tracker instance. Whether two trackers refer to the same `Suite` can be found out by calling `getSuite()` for both and compare the returned `Suite` instances. The tracker instance created for a management operation is passed to any call of `SuiteListener.notifySuiteStateChanged()` in order to inform about the progress of this operation.

For the installation of a new `Suite`, as long as the installation hasn't been successfully completed, an instance of `SuiteManagementTracker` is not assigned to any `Suite` instance yet, as it does not exit yet. In these cases, a call to `getSuite()` returns `null`.

In case of an update, the tracker is assigned to the existing `Suite` from the beginning, though.

This class has one method.

- `Suite getSuite()`

  This method returns the `Suite` that this tracker is assigned to, if the installation has completed successfully

SWMPermission Class

The `SWMPermission` provides permission handling for SWM API permissions. An `SWMPermission` object contains a scope and actions. The scope is the scope of the permission. Valid scopes are "`client`" stands for permission to perform the listed actions only for applications assigned to the same Client. "`crossClient`" stands for permission to perform the listed actions also for applications assigned to other Clients. Usually this is a permission reserved for the Root Client. Granting this permissions to other Clients should be figured out well in order to avoid security breaches.The actions to be granted are passed to the constructor in a non-empty string, containing a list of comma-separated keywords. Trailing and leading white spaces as well as those between the keywords and commas in the list are not allowed and lead to an `IllegalArgumentException`. The possible values can be seen in this table in the Security Policy Provider chapter of the spec. The actions string is converted to lowercase before processing.

This class has one constructor and several methods.

- `SWMPermission(java.lang.String scope, java.lang.String actions)`

  This method creates a new `SWMPermission` object with the specified name and actions.

- `public boolean implies(java.security.Permission p)`

  This method checks if the specified permission is "implied" by this object.

- `String getActions()`

  This method returns the permitted actions of this `Permission` as a comma separated list in alphabetical order.

- `java.security.PermissionCollection newPermissionCollection()`

  This method creates a new `SWMPermissionCollection`.

Task Class

The `Task` class is, in effect, a simple task descriptor. A `Task` is the abstraction of the execution of an application (see `javax.microedition.midlet.MIDlet`). Tasks are started using the `TaskManager.startTask()` method, where the arguments specify the application suite and the class within the suite being the starting point of the application. Starting a new task attempts to execute corresponding application. A task has a status, as described in the `TaskStatus` enumeration, that describes corresponding application lifecycle state. A task has a priority with possible values as described in `TaskPriority`. Depending on whether the implementation supports multiple VMs, several tasks can run in parallel.There are special tasks called system tasks. Those tasks cannot be started or stopped via this API, but are started by the

system. The `isSystemTask()` method can be used to find out whether a task is a considered a system task.

The `Task` class contains the following methods.

- `String getName()`

  This is a convenience method for returning the name of the task. The returned string is the name of the application running in this task.

- `TaskPriority getPriority()`

  This method returns the priority of given task.

- `public int getHeapUse()`

  This method returns the heap use of given task.

- `public TaskStatus getStatus()`

  This method returns the task's status.

- `public Suite getSuite()`

  This method returns the suite information this task executed from.

- `public boolean isSystemTask()`

  This method returns a boolean indicating whether a task is a system task.

InstallerErrorCode

The `InstallerErrorCode` provides several constants used by the installation routines. These constants are shown in Table 4-4.

Table 4-4 Installer Error Codes

Constant	Description
`ALREADY_INSTALLED`	The JAD matches a version of a suite already installed.
`APP_INTEGRITY_FAILURE_DEPENDENCY_CONFLICT`	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`	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`	Application Integrity Failure: hash mismatch.
`ATTRIBUTE_MISMATCH`	A attribute in both the JAD and JAR manifest does not match.
`AUTHORIZATION_FAILURE`	Application authorization failure, possibly indicating that the application was not digitally signed.

Table 4-4 (Cont.) Installer Error Codes

Constant	Description
CA_DISABLED	Indicates that the trusted certificate authority (CA) for this suite has been disabled for software authorization.
CANCELED	Canceled by user.
CANNOT_AUTH	The server does not support basic authentication.
CIRCULAR_COMPONENT_DEPENDENCY	Circular dynamic component dependency.
COMPONENT_DEPS_LIMIT_EXCEEDED	Dynamic component dependencies limit exceeded.
CONTENT_HANDLER_CONFLICT	The installation of a content handler would conflict with an already installed handler.
CORRUPT_DEPENDENCY_HASH	A dependency has a corrupt hash code.
CORRUPT_JAR	An entry could not be read from the JAR.
CORRUPT_PROVIDER_CERT	The content provider certificate cannot be decoded.
CORRUPT_SIGNATURE	The JAR signature cannot be decoded.
DEVICE_INCOMPATIBLE	The device does not support either the configuration or profile in the JAD.
DUPLICATED_KEY	Duplicated JAD/manifest key attribute
EXPIRED_CA_KEY	The certificate authority's public key has expired.
EXPIRED_PROVIDER_CERT	The content provider certificate has expired.
INSUFFICIENT_STORAGE	Not enough storage for this suite to be installed.
INVALID_CONTENT_HANDLER	The `MicroEdition-Handler-`<*n*> JAD attribute has invalid values.
INVALID_JAD_TYPE	The server did not have a resource with the correct type or the JAD downloaded has the wrong media type.
INVALID_JAD_URL	The JAD URL is invalid.
INVALID_JAR_TYPE	The server did not have a resource with the correct type or the JAR downloaded has the wrong media type.
INVALID_JAR_URL	The JAR URL is invalid.
INVALID_KEY	A key for an attribute is not formatted correctly.

Table 4-4 (Cont.) Installer Error Codes

Constant	Description
INVALID_NATIVE_LIBRARY	A native library contained within the JAR cannot be loaded.
INVALID_PAYMENT_INFO	Indicates that the payment information provided with the IMlet suite is incomplete or incorrect.
INVALID_PROVIDER_CERT	The signature of the content provider certificate is invalid.
INVALID_SIGNATURE	The signature of the JAR is invalid.
INVALID_VALUE	A value for an attribute is not formatted correctly.
INVALID_VERSION	The format of the version is invalid.
JAD_MOVED	The JAD URL for an installed suite is different than the original JAD URL.
JAD_NOT_FOUND	The JAD was not found.
JAD_SERVER_NOT_FOUND	The server for the JAD was not found.
JAR_CLASSES_VERIFICATION_FAILED	Not all classes within JAR package can be successfully verified with class verifier.
JAR_IS_LOCKED	Component or MIDlet or IMlet suite is locked by the system.
JAR_NOT_FOUND	The JAR was not found at the URL given in the JAD.
JAR_SERVER_NOT_FOUND	The server for the JAR was not found at the URL given in the JAD.
JAR_SIZE_MISMATCH	The JAR downloaded was not the same size as given in the JAD.
MISSING_CONFIGURATION	The configuration is missing from the manifest.
MISSING_DEPENDENCY_HASH	A dependency hash code is missing.
MISSING_DEPENDENCY_JAD_URL	A dependency JAD URL is missing.
MISSING_JAR_SIZE	The JAR size is missing.
MISSING_JAR_URL	The URL for the JAR is missing.
MISSING_PROFILE	The profile is missing from the manifest.
MISSING_PROVIDER_CERT	The content provider certificate is missing.
MISSING_SUITE_NAME	The name of MIDlet or IMlet suite is missing.
MISSING_VENDOR	The vendor is missing.

Table 4-4 (Cont.) Installer Error Codes

Constant	Description
MISSING_VERSION	The version is missing.
NEW_VERSION	This suite is newer that the one currently installed.
NO_ERROR	No error.
OLD_VERSION	This suite is older that the one currently installed.
OTHER_ERROR	Other errors.
PROXY_AUTH	Indicates that the user must first authenticate with the proxy.
PUSH_CLASS_FAILURE	The class in a push attribute is not in `MIDlet-<n>` attribute.
PUSH_DUP_FAILURE	The connection in a push entry is already taken.
PUSH_FORMAT_FAILURE	The format of a push attribute has an invalid format.
PUSH_PROTO_FAILURE	The connection in a push attribute is not supported.
REVOKED_CERT	The certificate has been revoked.
RMS_INITIALIZATION_FAILURE	Failure to import RMS data.
SUITE_NAME_MISMATCH	The MIDlet or IMlet suite name does not match the one in the JAR manifest.
TOO_MANY_PROPS	Indicates that either the JAD or manifest has too many properties to fit into memory.
TRUSTED_OVERWRITE_FAILURE	Indicates that the user tried to overwrite a trusted suite with an untrusted suite during an update.
UNAUTHORIZED	Web server authentication failed or is required.
UNKNOWN_CA	The certificate authority (CA) that issued the content provider certificate is unknown.
UNKNOWN_CERT_STATUS	The certificate is unknown to OCSP server.
UNSUPPORTED_CERT	The content provider certificate has an unsupported version.
UNSUPPORTED_CHAR_ENCODING	Indicates that the character encoding specified in the MIME type is not supported.
UNSUPPORTED_PAYMENT_INFO	Indicates that the payment information provided with the MIDlet or IMlet suite is incompatible with the current implementation.

Table 4-4 (Cont.) Installer Error Codes

Constant	Description
`UNTRUSTED_PAYMENT_SUITE`	Indicates that the MIDlet or IMlet suite has payment provisioning information but it is not trusted.
`VENDOR_MISMATCH`	The vendor does not match the one in the JAR manifest.
`VERSION_MISMATCH`	The version does not match the one in the JAR manifest.

5

General Purpose Input/Output

This chapter describes the General Purpose Input/Output (GPIO) functionality in the Oracle Java ME Embedded product. GPIO typically refers a generic pin on an embedded board whose behavior, including whether it is an input or output pin, can be programmed by the user at runtime.

GPIO pins are often lined up in rows. By design, they have no dedicated purpose, and are used by programmers for a wide variety of tasks. For example:

- GPIO pins can be *enabled* or *disabled*.
- GPIO pins can be configured to be *input* or *output*.
- Input values are readable, often with a 1 representing a high voltage, and a 0 representing a low voltage.
- Input GPIO pins can be used as "interrupt" lines, which allow a peripheral board connected via multiple pins to signal to the primary embedded board that it requires attention.
- Output pin values are both readable and writable.

Warning:

Be sure to observe manufacturer's specifications and warnings carefully. For example, with the Raspberry Pi board, the voltage value that represents a "high" value on an input pin may be 3.3 volts (+3.3V). However, other pins may output 5 volts (+5V). Be sure to check the manufacturer's specifications to ensure that you are not placing too much voltage on an input GPIO line, as the board may not have an overvoltage protection.

GPIO pins have much greater functionality than this, but it is important to start with the basics.

Setting a GPIO Output Pin

For this example, you will need the following hardware:

Table 5-1 Hardware for GPIO Example

Hardware	Where to Obtain
Raspberry Pi 512 MB Rev B, B+, or Raspberry Pi 2	Various third-party sellers
Multimeter	Various. Sinometer DT830B used in the example.

Perhaps the simplest example of working with the GPIO functionality in the Oracle Java ME Embedded product is to set the high/low value of an arbitrary output pin and read its voltage with a multimeter. In this example, we set the value of GPIO pin 7 to alternate between high (3.3V) and low (0V) at intervals of 10 seconds and 5 seconds, respectively. The following example shows the source code.

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

public class GPIOExample1 extends MIDlet {

    GPIOPin pin;

    public void startApp() {

        try {

            pin = (GPIOPin) DeviceManager.open(7);
            System.out.println("----------------------------------------");
            Thread.sleep(5000);

            for (int i = 0; i < 20; i++) {
                System.out.println("Setting pin to true...");
                pin.setValue(true);
                Thread.sleep(10000);
                System.out.println("Setting pin to false...");
                pin.setValue(false);
                Thread.sleep(5000);
                System.out.println("----------------------------------------");

            }

        } catch (IOException ex) {
            Logger.getLogger(GPIOExample1.class.getName()).
                log(Level.SEVERE, null, ex);
        } catch (InterruptedException ex) {
            Logger.getLogger(GPIOExample1.class.getName()).
                log(Level.SEVERE, null, ex);
        }
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {

        try {
            pin.close();
        } catch (IOException ex) {
            Logger.getLogger(GPIOExample1.class.getName()).
                log(Level.SEVERE, null, ex);
        }

    }
}
```

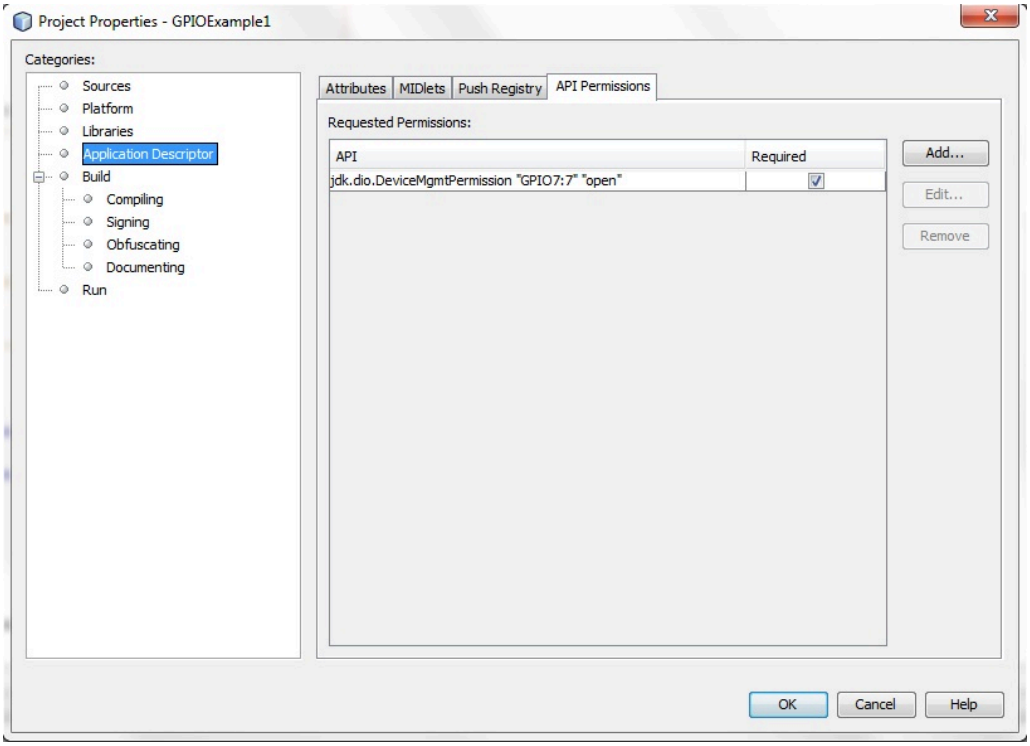
The following permissions must be added to the Application Descriptor of the IMlet so that it will execute without any security exceptions from the Oracle Java ME Embedded runtime.

Table 5-2 Permissions for GPIO Example

Permission	Device	Operation
`jdk.dio.DeviceMgmtPermission`	`GPIO7:7`	`open`

Note that if you're using an IDE such as NetBeans as the development environment, you will need to access the project properties of the project and set API permissions under the application descriptor, as shown in Figure 5-1.

Figure 5-1 API Permissions in the Application Descriptor in NetBeans



After running the application, set your multimeter to read DC voltage with a maximum of 20V, then connect one of the leads of the multimeter to GPIO 7, and the other to GND (ground). As the application is running, note that the voltage that is read by the multimeter will jump from its low value of around 0V after a call to `pin.setValue(false)` to its high value of around 3.3V after a call to `pin.setValue(true)`. This is shown in Figure 5-2 and Figure 5-3.

Warning:

Remember that the GPIO pin assignments on the Raspberry Pi do *not* match the pin numbers on the board. For example, GPIO 7 is *not* mapped to pin 7, but instead pin 26. See Appendix A (or the hardware-appropriate Getting Started Guide) for the pin assignments for the target boards of the Oracle Java ME Embedded software.

Figure 5-2 Raspberry Pi Pin 7 with Low (0V) Voltage

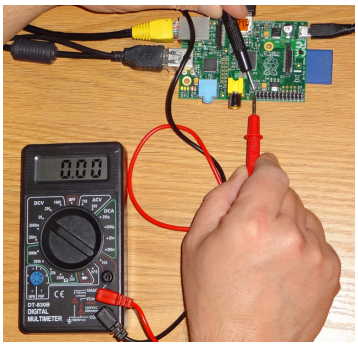
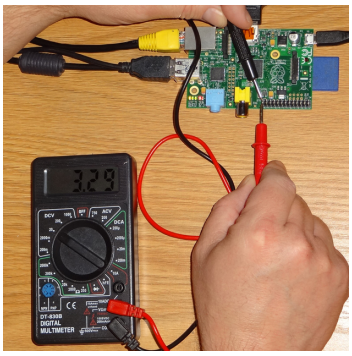


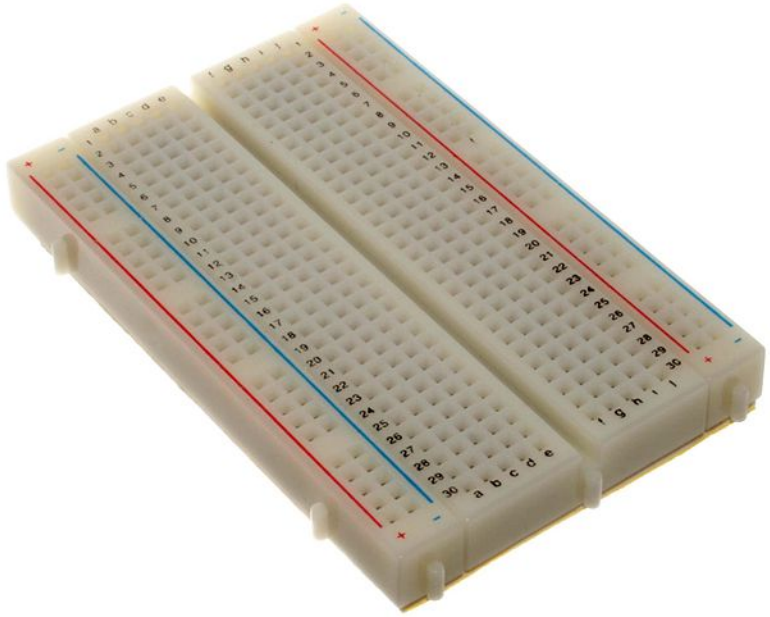
Figure 5-3 Raspberry Pi Pin 7 with High (3.3V) Voltage



Working with a Breadboard

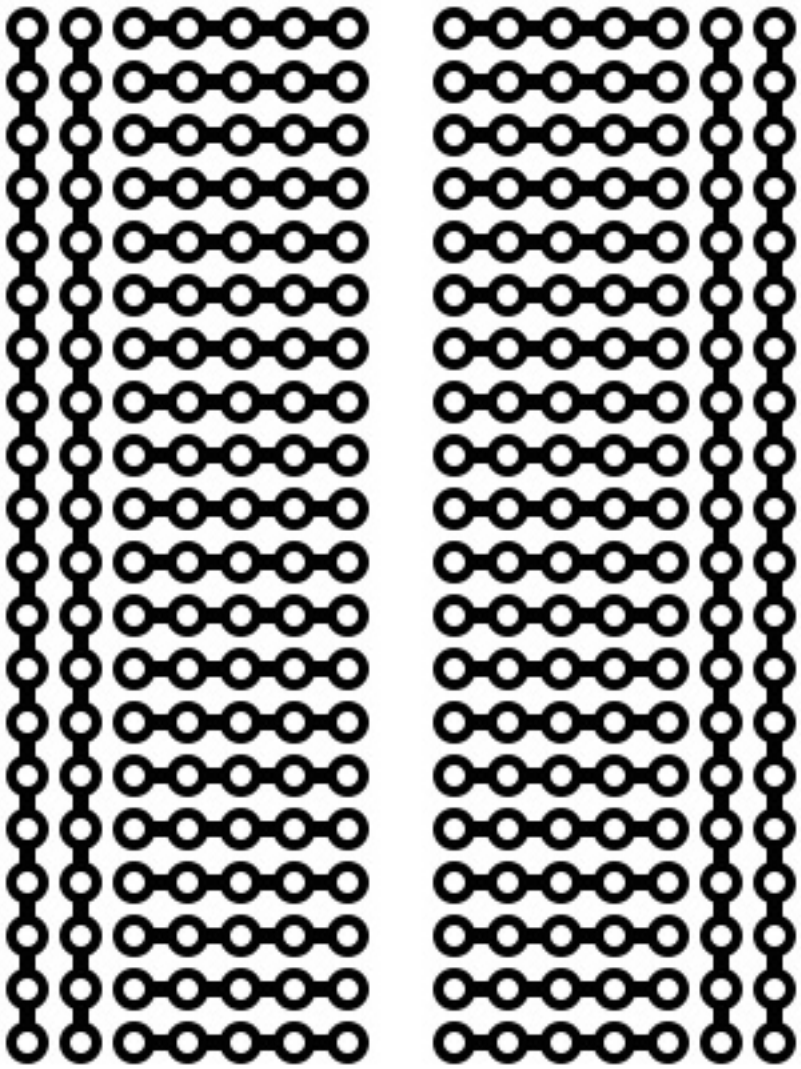
When prototyping circuits, it is often helpful to have a way of connecting wires without having to perform soldering. In some cases, if there are only a few connections, you can use jumper wires. However, when layout out more complex circuits, it's helpful to use a breadboard. A typical breadboard is shown in Figure 5-4.

Figure 5-4 A Typical Breadboard



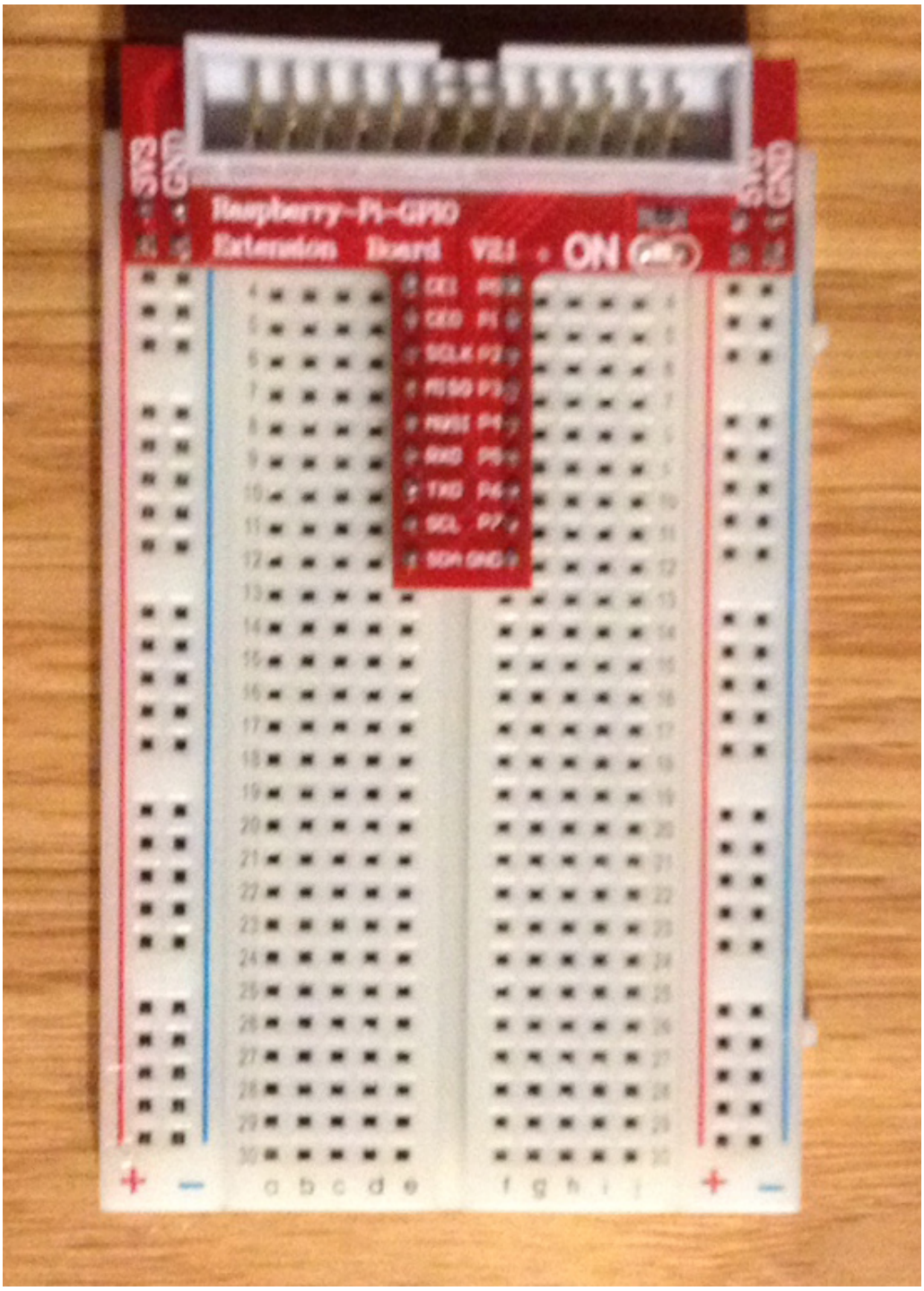
A breadboard consists of a large number of holes, each of which are wired together on the bottom using a standardized pattern, such as the one shown in Figure 5-5. Note that the two columns on both the left and the right of the breadboard are wired vertically--these provide power (+) and ground (-) connections that can be tapped into to. The horizontal rows on either side of the center line, on the other hand, are used to create circuits. Circuits can be created using small wires with metal tips on each end that can "plug into" the holes.

Figure 5-5 Wiring Pattern for a Typical Breadboard



For the Raspberry Pi, we can connect the GPIO pins on the Pi to a breadboard using a device called a T-Cobbler Extension Board. This device attaches a ribbon cable to the GPIO pins, which in turn connects to the T-cobbler board. The T-cobbler board is then inserted into the top of the breadboard, as shown in Figure 5-6.

Figure 5-6 T-Cobbler Extension Board for the Raspberry Pi



Once connected to the Pi, you can use any of the holes running along the red stripe on the left side of the breadboard to provide +3.3 volts (3V3), or any of the holes running along the red stripe on the right side of the breadboard to provide +5 volts (5V0). In addition, any of the holes running along the blue stripes on either side of the board connect to the ground (GND) on the Raspberry Pi.

The GPIO pins on the Raspberry Pi map to the pins on the T-cobbler (and hence the respective horizontal rows on the breadboard) as shown in Figure 5-3.

Table 5-3 Broadcom GPIO to T-Cobbler Conversion

GPIO (Pi Pin Number)	Alternate Name
2 (Pin 3)	SDA
3 (Pin 5)	SCL
4 (Pin 7)	P7
7 (Pin 26)	CE1
8 (Pin 24)	CE0
9 (Pin 21)	MISO
10 (Pin 19)	MOSI
11 (Pin 23)	SCLK
14 (Pin 8)	TXD
15 (Pin 10)	RXD
18 (Pin 12)	P1
22 (Pin 15)	P3
23 (Pin 16)	P4
24 (Pin 18)	P5
25 (Pin 22)	P6
27 (Pin 13)	P2

Blinking an LED

We can use the code in the first example to create a small circuit on the breadboard that turns on an off a light-emitting diode (LED). For this example, you will need the following equipment.

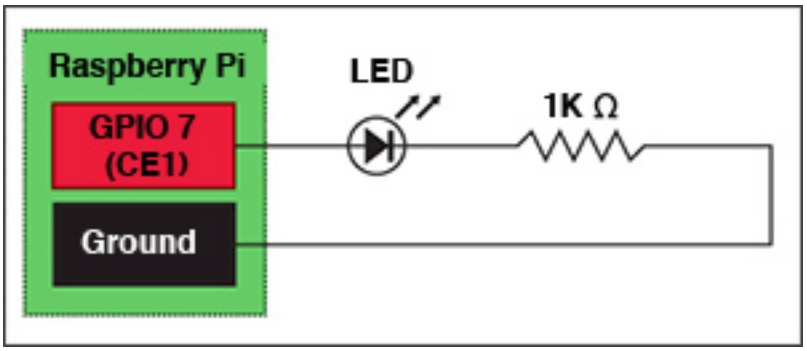
Table 5-4 Equipment Needed for Blinking LED Example

Hardware	Where to Obtain
LED	Any electronics store
1000 ohm resistor	Any electronics store
T-Cobbler and Breadboard	Adafruit
Jumper Wires (Male to Male)	Adafruit

Use the breadboard to connect one end of a 1000-ohm resistor to a row that connects to GPIO7, which is marked on the T-Cobbler by CE1. Plug the other end of the 1000-ohm

resistor into an unused row further down the breadboard. Then, run an LED from that row an adjacent row, and then connect that row to the ground (GND). The circuit should look similar to the schematic in Figure 5-7.

Figure 5-7 Schematic for Wiring an LED to GPIO 7

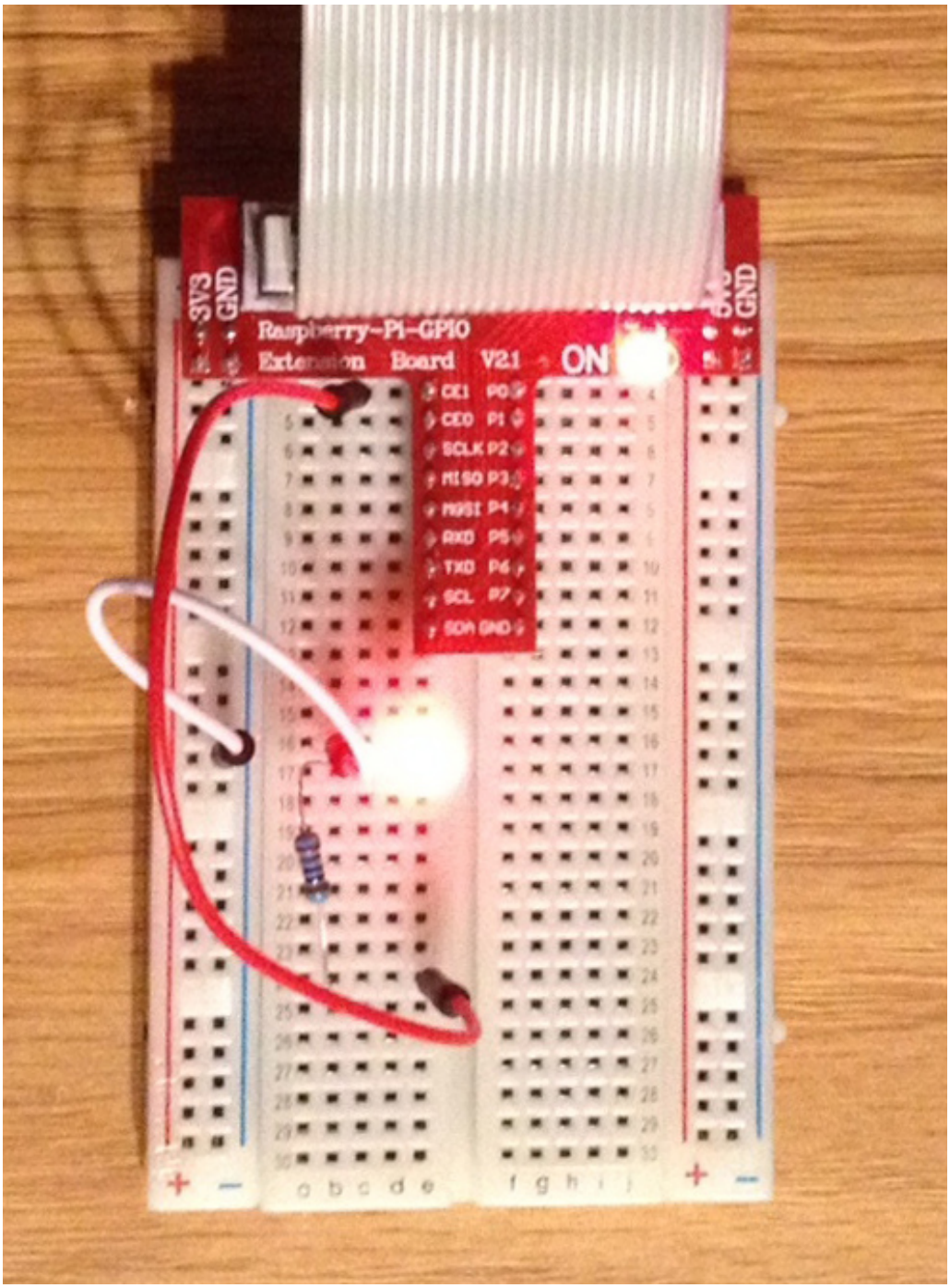


When completed, you should have a prototype that looks like Figure 5-8. Run the first example again, and you should see the LED light blinking off an on whenever the `setValue(true)` call is made on the `GPIOPin` object.

Note:

Remember that an LED is a *diode*, which by definition only allows current to travel one way through it. If your LED does not light up when the voltage is applied, try flipping the connections so that the current travels the reverse direction through the diode.

Figure 5-8 Wiring an LED to GPIO Pin 7



Testing Output and Input Pins

Our next GPIO example will take the output voltage from one pin and redirect it back to an adjacent input pin, while creating a listener on the input pin that reacts accordingly. For this example, you will need the following hardware:

Table 5-5 Hardware for Example 1-1

Hardware	Where to Obtain
Raspberry Pi 512 MB Rev B	Various third-party sellers
Multimeter	Various. Sinometer DT830B used in the example.

Here, we use GPIO 8 and 11 on the Raspberry Pi due to their proximity to each other. These pins are right next to GPIO 7 and GND, which was used in the previous example. In the example below, we've added a listener to an input pin that will trigger whenever the input voltage changes in both directions (high-to-low and low-to-high).

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

public class GPIOExample2 extends MIDlet {

    GPIOPin pin8;
    GPIOPin pin11;

    public void startApp() {

        try {

            pin8 = (GPIOPin) DeviceManager.open(8);   //  Output pin by default
            pin11 = (GPIOPin) DeviceManager.open(11);  //  Input pin by default
            pin11.setInputListener(new MyPinListener());

            System.out.println("----------------------------------------");
            Thread.sleep(5000);

            for (int i = 0; i < 20; i++) {
                System.out.println("Setting pin 8 to true...");
                pin8.setValue(true);
                Thread.sleep(10000);
                System.out.println("Setting pin 8 to false...");
                pin8.setValue(false);
                Thread.sleep(5000);
                System.out.println("----------------------------------------");

            }

        } catch (IOException ex) {
            Logger.getLogger(GPIOExample2.class.getName()).
                log(Level.SEVERE, null, ex);
        } catch (InterruptedException ex) {
            Logger.getLogger(GPIOExample2.class.getName()).
                log(Level.SEVERE, null, ex);
        }
    }
```

```
    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {

        try {
            pin8.close();
            pin11.close();
        } catch (IOException ex) {
            Logger.getLogger(GPIOExample2.class.getName()).
                log(Level.SEVERE, null, ex);
        }

    }

    class MyPinListener implements PinListener {

        @Override
        public void valueChanged(PinEvent event) {
            try {
                System.out.println("Pin listener for pin 11 has been called!");
                System.out.println("Pin 11 is now " + pin11.getValue());
            } catch (IOException ex) {
                Logger.getLogger(GPIOExample2.class.getName()).
                    log(Level.SEVERE, null, ex);
            }
        }

    }
}
```

Table 5-6 shows the permission that must be added to the Application Descriptor of the IMlet so that it will execute without any security exceptions from the Oracle Java ME Embedded runtime.

Table 5-6 Permissions for Example 1-2

Permission	Device	Operation
`jdk.dio.DeviceMgmtPermission`	`*:*`	`open`

After running the application, either connect one of the leads of the multimeter to the GPIO 8 pin and the other to the GPIO 11 pin of the Raspberry Pi (or create a compatible circuit on a breadboard). Set your multimeter to read DCV with a maximum of 200 mV. As the application is running, note that the voltage that is read by the multimeter will jump from its low value to its high voltage, although the voltages will be much smaller than that from GPIO 7. Try disconnecting the lead from GPIO 11 momentarily and reconnecting it when GPIO 8 is high. The output of the program should reflect that the listener is called both when the lead is released, and when it is reconnected.

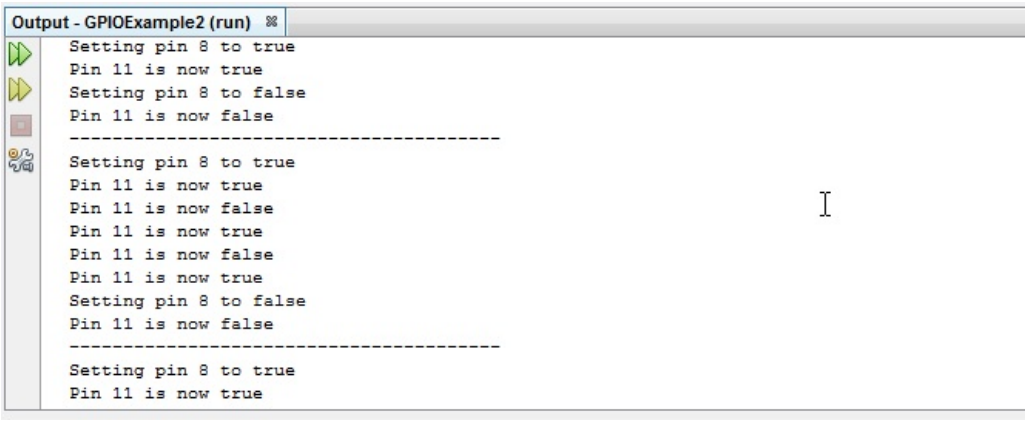
Warning:

Remember that the GPIO pin assignments on the Raspberry Pi do *not* match the pin numbers on the board. For example, GPIO 8 is *not* mapped to pin 8, but instead pin 24. Likewise, GPIO 11 is mapped to pin 23. See Appendix A

and Appendix B for the pin assignments for the target boards of the Oracle Java ME Embedded software.

The output of the application when running in NetBeans is shown in Figure 5-9.

Figure 5-9 Output of Example 1-2



6

Working with the I2C Bus

The I2C bus, often referred to as "i-2-c" or "i-squared-c", is a low-speed bus frequently used between micro-controllers and peripherals. I2C uses only two bi-directional lines, Serial Data Line (SDA) and Serial Clock (SCL), often pulled-up with resistors. Typical voltages used are +5 V or +3.3 V, although systems with other voltages are permitted.

When using the Raspberry Pi, be sure to check the manufacturer's specifications as to which voltages are acceptable for powering the peripheral. The Raspberry Pi provides both 3.3V and 5V pins.

To enable I2C on the Raspberry Pi, add the following lines to the `/etc/modules` files and reboot. Note that the file will need to be edited with root privileges.

```
i2c-bcm2708
i2c-dev
```

Experimenting with a 7-Segment Display

For this exercise, you will need the following hardware:

Table 6-1 Hardware for 7-Segment Display Example

Hardware	Where to Obtain
Raspberry Pi 512 MB Rev B, B+, or Raspberry Pi 2	Various third-party sellers
Adafruit .56" 4-digit 7-segment display with HT16K33 I2C Backpack	Adafuit or Amazon. Requires a small amount of soldering of the LED display unit to I2C logic board, as well as 4 I2C connector pins.
Jumper Wires - Female to Female (x4)	Electronics store. We used SchmartBoard P/N 920-0065-01 Rev A

Our first example allows us to use the GPIO2 and GPIO3 pins for the I2C data and clock connections. Using these connections, we will write a simple program that allows us to set the display using an I2C connection.

In order to hook up the 7-Segment display to the Raspberry Pi properly, the jumper wires must be connected as shown in Table 6-2. Note that because there are only four connections, we opted not to use a T-cobber and a breadboard in this example.

Table 6-2 Raspberry Pi to HT16K33 Jumper Connections

Pins on Raspberry Pi	HT16K33 Board
5V (Pin 2)	VCC

Table 6-2 (Cont.) Raspberry Pi to HT16K33 Jumper Connections

Pins on Raspberry Pi	HT16K33 Board
Ground (Pin 6)	GND
GPIO 2 (Pin 3)	SDA (Serial Data)
GPIO 3 (Pin 5)	SCL (Serial Clock)

First, we need a basic class that communicates with the HT16K33 "LED backpack" that is soldered to the actual 7-segment LED display. The following example shows the source code for the 7-segment I2C display driver.

```
import jdk.dio.DeviceManager;
import jdk.dio.i2cbus.I2CDevice;
import jdk.dio.i2cbus.I2CDeviceConfig;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.util.logging.Level;
import java.util.logging.Logger;

public class LEDBackpack {

    I2CDeviceConfig LEDBackpackConfig;
    int[] displaybuffer = new int[10];

    byte[] OSCILLATOR_ON = {0x21};
    byte BRIGHTNESS = (byte) 0xE0;

    static byte HT16K33_BLINK_CMD = (byte) 0x80;
    static byte HT16K33_BLINK_DISPLAYON = (byte) 0x01;

    static byte HT16K33_BLINK_OFF = (byte) 0;
    static byte HT16K33_BLINK_2HZ = (byte) 1;
    static byte HT16K33_BLINK_1HZ = (byte) 2;
    static byte HT16K33_BLINK_HALFHZ = (byte) 3;

    static byte LETTER_J = 0x1E;
    static byte LETTER_A = 0x77;
    static byte LETTER_V = 0x3E;

    static final byte numbertable[] = {
        0x3F, /* 0 */
        0x06, /* 1 */
        0x5B, /* 2 */
        0x4F, /* 3 */
        0x66, /* 4 */
        0x6D, /* 5 */
        0x7D, /* 6 */
        0x07, /* 7 */
        0x7F, /* 8 */
        0x6F, /* 9 */
        0x77, /* a */
        0x7C, /* b */
        0x39, /* C */
        0x5E, /* d */
        0x79, /* E */
```

```
    0x71, /* F */};

public LEDBackpack() {
    LEDBackpackConfig = new I2CDeviceConfig(1, 0x70, 7, 100000);
}

void begin() {

    try (I2CDevice slave = DeviceManager.open(LEDBackpackConfig)) {

        ByteBuffer oscOnCmd = ByteBuffer.wrap(OSCILLATOR_ON);
        slave.write(oscOnCmd);
        slave.close();

    } catch (IOException ioe) {
        Logger.getLogger(LEDBackpack.class.getName()).
        log(Level.SEVERE, null, ioe);
    }

    setBlinkRate(HT16K33_BLINK_OFF);
    setBrightness(15);

}

void setBrightness(int b) {

    if (b > 15) {
        b = 15;
    } else if (b < 0) {
        b = 0;
    }

    byte[] ea = {(byte) (BRIGHTNESS | b)};

    try (I2CDevice slave = DeviceManager.open(LEDBackpackConfig)) {

        ByteBuffer brightnessCmd = ByteBuffer.wrap(ea);
        slave.write(brightnessCmd);
        slave.close();

    } catch (IOException ioe) {
        Logger.getLogger(LEDBackpack.class.getName()).
            log(Level.SEVERE, null, ioe);
    }
}

void setBlinkRate(int b) {

    if (b > 3) {
        b = 0; // turn off if not sure
    } else if (b < 0) {
        b = 0;
    }

    byte[] ea =
        {(byte) (HT16K33_BLINK_CMD | HT16K33_BLINK_DISPLAYON | (b << 1))};

    try (I2CDevice slave = DeviceManager.open(LEDBackpackConfig)) {

        ByteBuffer blinkRateCmd = ByteBuffer.wrap(ea);
        slave.write(blinkRateCmd);
```

```
            slave.close();

        } catch (IOException ioe) {
            Logger.getLogger(LEDBackpack.class.getName()).
                log(Level.SEVERE, null, ioe);
        }
    }

    void writeDisplay() {

        try (I2CDevice slave = DeviceManager.open(LEDBackpackConfig)) {

            byte start[] = {0x00};

            ByteBuffer startCmd = ByteBuffer.wrap(start);
            slave.write(0x00, 1, startCmd);

            for (int i = 0; i < displaybuffer.length; i++) {

                byte b1a[] = {(byte) (displaybuffer[i] & 0xFF)};
                ByteBuffer b1Cmd = ByteBuffer.wrap(b1a);
                slave.write(i, 1, b1Cmd);

            }

            slave.close();

        } catch (IOException ioe) {
            Logger.getLogger(LEDBackpack.class.getName()).
                log(Level.SEVERE, null, ioe);
        }

    }

    void clear() {
        for (int i = 0; i < displaybuffer.length; i++) {
            displaybuffer[i] = 0;
        }
    }

}
```

This driver class contains five methods: `begin()`, `setBrightness()`, `setBlinkRate()`, `writeDisplay()`, and `clear()`. Let's cover each of these in more detail.

The `begin()` method will initialize the display. There are three operations that must be performed to do this properly. First, the oscillator on the HT16K33 LED backboard must be turned on. We can do this by sending a byte value of hex 0x21 across the bus. Next, we set the blink rate of the 7-segment display to one of four values: OFF, 2 Hz, 1 Hz, or .5 Hz. Finally, we can set the brightness of the display using a value of 1 to 15. For the latter two operations, we make use of the next two methods which can also be called independently.

The `setBlinkRate()` and `setBrightness()` methods simply take an input value, perform bounds checking, and calculate the correct byte value to send across the bus. Just like turning on the oscillator, we only need to send one byte across the bus to modify the blink rate or brightness to any level we choose.

The `writeDisplay()` method, on the other hand, is a little more complex. Here, the class makes use of an array of 10 integers, declared as a field, that serves as a display

buffer. In reality, the `writeDisplay()` method will truncate any value larger then 255 before sending it across the bus, but making it an array of integers is helpful for the user.

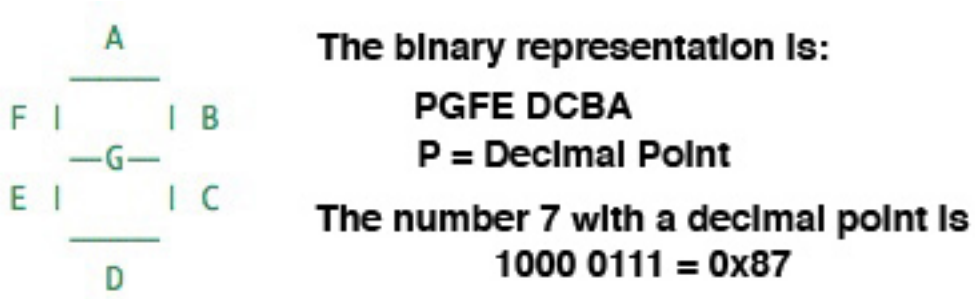
Each of the entries in the array will map to an address on the HT16K33 "LED backpack" that can be written to using the I2C bus. The purpose of each of the addresses is shown in Table 6-3. Note that since the HT16K33 can drive different types of LED displays, several of the addresses are ignored when using this particular 4-character 7-segment display.

Table 6-3 HT16K33 7-Segment Display Addresses

Address	Purpose
0x00	7-Segment Display Character 1 and Period
0x01	Ignored
0x02	7-Segment Display Character 2 and Period
0x03	Ignored
0x04	Colon (0xFF for colon on; 0x00 for colon off)
0x05	Ignored
0x06	7-Segment Display Character 3 and Period
0x07	Ignored
0x08	7-Segment Display Character 4 and Period
0x09	Ignored

Each address can have one byte written to it. The contents of each byte is mapped out in binary as shown in Figure 6-1. As such, the number 7 with a decimal point is represented in binary as 10000111, which is equal to 0x87 in hexadecimal. Note that address 0x04 is reserved for the colon that appears between the first two numbers and the second two numbers in the display; it does not represent character 3.

Figure 6-1 Binary Encoding for 7-Segment Display



The following example shows a sample IMlet that will write the word "JAVA", without any decimal points or colon, to the display (even though the "V" looks the same as a "U" in the 7-segment display).

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

```
public class I2CExample1 extends MIDlet {

    public void startApp() {

        LEDBackpack backpack = new LEDBackpack();

        backpack.begin();
        backpack.setBrightness(10);
        backpack.setBlinkRate(LEDBackpack.HT16K33_BLINK_OFF);

        backpack.clear();
        backpack.writeDisplay();

        backpack.displaybuffer[0] = LEDBackpack.LETTER_J;
        backpack.displaybuffer[2] = LEDBackpack.LETTER_A;
        backpack.displaybuffer[4] = 0x00;                          //  No colon
        backpack.displaybuffer[6] = LEDBackpack.LETTER_V;
        backpack.displaybuffer[8] = LEDBackpack.LETTER_A;
        backpack.writeDisplay();

    }

    public void pauseApp() {
    }

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

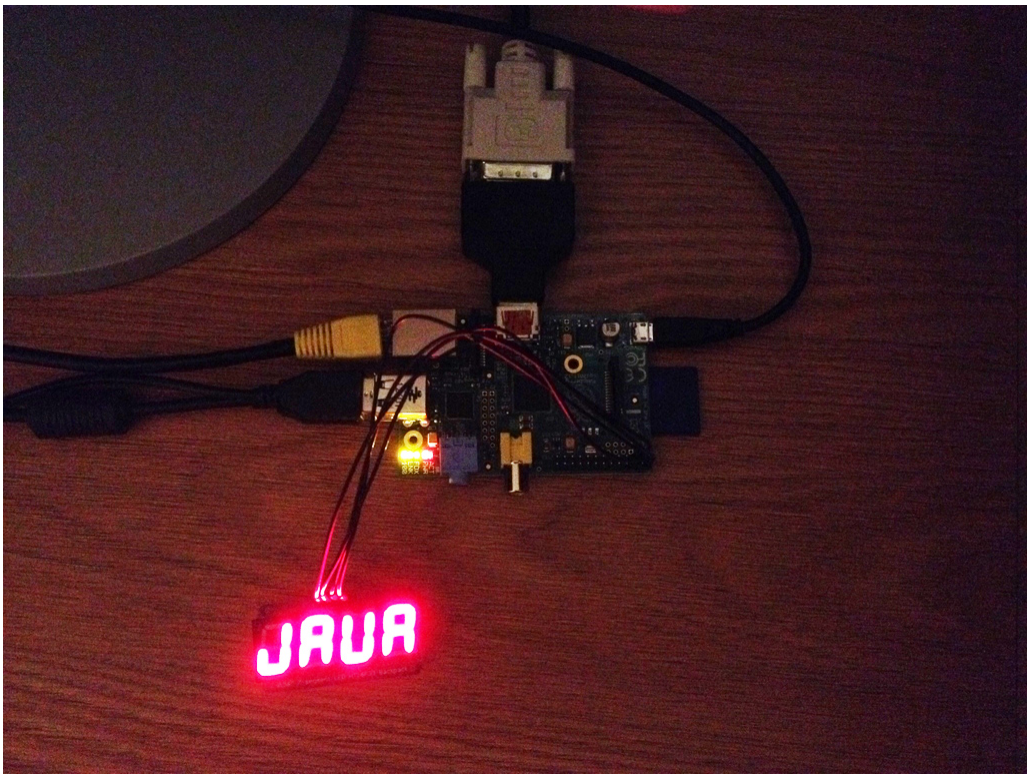
The following permissions must be added to the Application Descriptor of the project so that it will execute without any security exceptions from the Oracle Java ME Embedded runtime.

Table 6-4 API Permissions for 7-Segment Display Project

Permission	Device	Operation
`jdk.dio.DeviceMgmtPermission`	`*:*`	`open`
`jdk.dio.i2cbus.I2CPinPermission`	`*:*`	`open`

After running the application, you should see the display as shown in Figure 6-2.

Figure 6-2 Result of Running the 7-Segment Display IMlet



Experimenting with a 16x2 LCD Display

For this exercise, you will need the following hardware:

Table 6-5 Hardware for Example 2-2

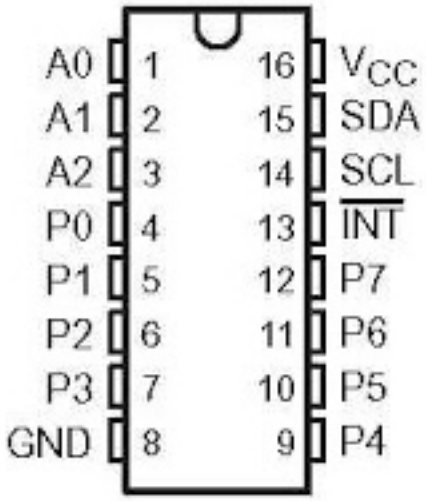
Hardware	Where to Obtain
Raspberry Pi 512 MB Rev B, B+, or Raspberry Pi 2	Various third-party sellers
16x2 LCD Display with an HD44780 Controller	Amazon. Requires a small amount of soldering for the 16 connector pins that run on the top of the logic board.
PCF8574N 8-bit I/O Expander Chip	Mouser Electronics.
T-Cobbler and Breadboard	Electronics store.
Jumper Wires	Electronics store

This example uses the I2C bus to interface to an LCD display with a Hitachi HD44780 backboard. The HD44780-based 16x2 character LCDs are inexpensive and widely available. However, in addition to the LCD display, we must also use a PCF8574-based IC, which is an general purpose bidirectional 8 bit I/O port expander that uses the I2C protocol.

The first step is to hook up the Raspberry Pi to the PCF8574 chip. Typically, an IC chip is installed on a breadboard vertically along the center aisle, with the pins from the IC

connecting to the holes adjacent to the center. The pinouts for the PCF8574N IC are shown in Figure 6-3.

Figure 6-3 Pinout Diagram for PCF8574N IC



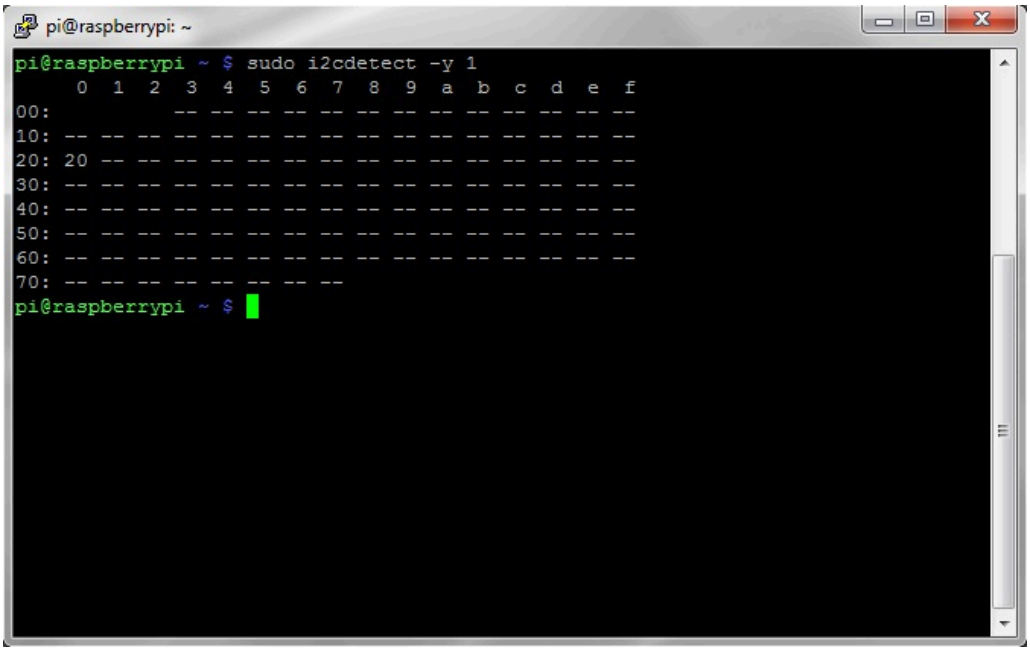
Once the chip is on the breadboard, there are several pins on the chip that must be connected to the T-Cobbler using jumper wires, as shown in Table 6-6.

Table 6-6 Raspberry Pi to PCF8574N Jumper Connections

Pins on T-Cobbler (Pi)	PCF8574N Pins
+5V (Pin 2)	VCC
GND (Pin 6)	GND
SDA / GPIO 2 (Pin 3)	SDA (Serial Data)
SCL / GPIO 3 (Pin 5)	SCL (Serial Clock)
GND (Pin 6)	A0
GND (Pin 6)	A1
GND (Pin 6)	A2

The first four pins shown in are the standard I2C connections that are required of any slave device that wishes to use the I2C bus. However, the remaining 3 pins are used to set the slave address on I2C bus #1, represented as a binary digit from 0-7 (A0=1, A1=2, A2=4) that is added to the hexidecimal value of 0x20. Because we are not running voltage on any of these pins, the address of the PCF8574N chip on the I2C bus should remain 0x20. If you'd like to verify this, login to the Raspberry Pi and issue the command shown in Figure 6-4. Here, the `i2cdetect` command shows that on bus 1 there is a device at address 0x20. To change the address, try connecting a 10K resistor between the 5V pin and one of the A*x* pins and rerunning the command. The address that is reported should change accordingly.

Figure 6-4 Running the i2cdetect Command



The remaining pins P0-P7 and INT (high) on the PCF8574N are used to communicate with other devices, in this case the HD44780 chip that drives the 16x2 LCD display. Table 6-7 shows the connections to and the PCF8574N chip.

Table 6-7 Connections to PCF8574N Chip

Raspberry Pi (T-Cobbler)	PCF8574N
SCL / GPIO 3 (Pin 5)	SCL
SDA / GPIO 2 (Pin 3)	SDA
GND (Pin 6)	A0 (see discussion on I2C address above)
GND (Pin 6)	A1
GND (Pin 6)	A2
+5V (Pin 2)	VDD
GND (Pin 6)	VSS

Table 6-8 shows the connections between the PCF8574N chip and the HD44780 controller.

Table 6-8 Connection Between PCF8574N Chip and HD44780 Controller

PCF8574N	HD44780
P0	DB4
P1	DB5
P2	DB6

Table 6-8 (Cont.) Connection Between PCF8574N Chip and HD44780 Controller

PCF8574N	HD44780
P3	DB7
P4	RS
P5	R/W
P6	E

Table 6-9 shows connection between the T-Cobbler and the HD44780 controller.

Table 6-9 Connection to HD44780 Chip

Raspberry Pi (T-Cobbler)	HD44780
+5V (Pin 2)	VDD
0 to +5V	VO (variable resistor if desired for dimming backlit display)
GND (Pin 6)	VSS

Before connecting the P*x* lines on the IC, try placing a resistor and an LED on a line coming from the P0 pin. Then, run the code shown in the following example.

```
import javax.microedition.midlet.MIDlet;
import jdk.dio.DeviceManager;
import jdk.dio.i2cbus.I2CDevice;
import jdk.dio.i2cbus.I2CDeviceConfig;
import java.io.IOException;

public class IOExpanderExample extends MIDlet {

    public void startApp() {

        LEDBackpackConfig = new I2CDeviceConfig(1, 0x20, 7, 100000);
        try (I2CDevice slave = DeviceManager.open(LEDBackpackConfig))
        {

            slave.write((byte)0x01);

        } catch (IOException ex) {
            //  Handle exception
        }

    }

    public void pauseApp() {

    }

    public void destroyApp(boolean unconditional) {

    }
```

```
}
```

To understand this example, it helps to look at the data line dialog, as shown in Figure 6-5. Each of the P*x* lines can be activated or deactivated by writing a binary number to the slave device, where P7 represents the most-significant digit and P0 represents the least-significant digit. Writing a value of 0x01 to the slave device will activate only the P0 line, which should in turn make the LED that is connected to it light up (be sure that the LED's cathode and anode connected are the right direction and that there is a resistor in line so the LED does not burn out!). Note that the LED will remain lit until a new value is written to the bus, or the PCF8574N chip loses power.

Figure 6-5 I/O Data Bus with the PCF8574N chip

BYTE	BIT							
	7 (MSB)	6	5	4	3	2	1	0 (LSB)
I/O data bus	P7	P6	P5	P4	P3	P2	P1	P0

Next, complete the circuit according to Table 6-7. The following shows a sample driver class that will control the HD44780.

```
import javax.microedition.midlet.MIDlet;
import jdk.dio.DeviceManager;
import jdk.dio.i2cbus.I2CDevice;
import jdk.dio.i2cbus.I2CDeviceConfig;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

public class LCDDisplay {

    I2CDeviceConfig LEDBackpackConfig;
    I2CDevice slave;

    public LCDDisplay()
            throws InterruptedException, IOException {

        LEDBackpackConfig = new I2CDeviceConfig(1, 0x20, 7, 100000);
        slave = DeviceManager.open(LEDBackpackConfig);

    }

    public void begin()
            throws InterruptedException, IOException {

        slave.write(0x03);
        byte result1 = (byte) slave.read();
        Thread.sleep(5);

        slave.write(0x03);
        byte result2 = (byte) slave.read();

        Thread.sleep(1);
        slave.write(0x03);
        byte result3 = (byte) slave.read();

        Thread.sleep(1);
```

```
        slave.write(0x02);
        byte result4 = (byte) slave.read();

        writeCommand((byte) 0x28);
        writeCommand((byte) 0x08);
        writeCommand((byte) 0x01);
        writeCommand((byte) 0x06);
        writeCommand((byte) 0x0C);

        Thread.sleep(1);
        byte result5 = (byte) slave.read();

    }

    public void writeCharacter(byte charvalue)
            throws InterruptedException, IOException {

        slave.write((byte) (0x10 | (charvalue >> 4)));
        strobe();
        slave.write((byte) (0x10 | (charvalue & 0x0F)));
        strobe();
        slave.write(0x00);
        Thread.sleep(1);

    }

    public void writeCommand(byte value)
            throws InterruptedException, IOException {

        slave.write((byte) (value >> 4));
        strobe();
        slave.write((byte) (value & 0x0F));
        strobe();
        slave.write(0x00);
        Thread.sleep(5);

    }

    public void writeString(int line, String string)
            throws InterruptedException, IOException {

        if (line == 1) {
            writeCommand((byte) 0x80);
        } else if (line == 2) {
            writeCommand((byte) 0xC0);
        } else if (line == 3) {
            writeCommand((byte) 0x94);
        } else if (line == 4) {
            writeCommand((byte) 0xD4);
        }

        char[] chars = string.toCharArray();

        for (int i = 0; i < chars.length; i++) {
            writeCharacter((byte) chars[i]);
        }
    }

    public void strobe()
            throws InterruptedException, IOException {
```

```
        Thread.sleep(1);

        byte readResult = (byte) slave.read();
        readResult |= 0x40;
        slave.write(readResult);

        Thread.sleep(1);

        readResult = (byte) slave.read();
        readResult &= 0xBF;
        slave.write(readResult);

    }

    public void clear()
            throws InterruptedException, IOException {

        Thread.sleep(5);

        writeCommand((byte) 0x01);
        Thread.sleep(5);

        writeCommand((byte) 0x02);
        Thread.sleep(5);

    }

    public void end()
            throws IOException {

        slave.close();

    }

}
```

To use the driver class, run the IMlet shown in the following example.

```
import java.io.IOException;
import javax.microedition.midlet.MIDlet;

public class I2CExample2 extends MIDlet {

    public void startApp() {

        LCDDisplay display;
        try {
            display = new LCDDisplay();
            display.begin();
            display.clear();
            display.writeString(1, "Java ME");
            display.writeString(2, "Embedded");
            display.end();
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
```

```
    }

    public void pauseApp() {
    }

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

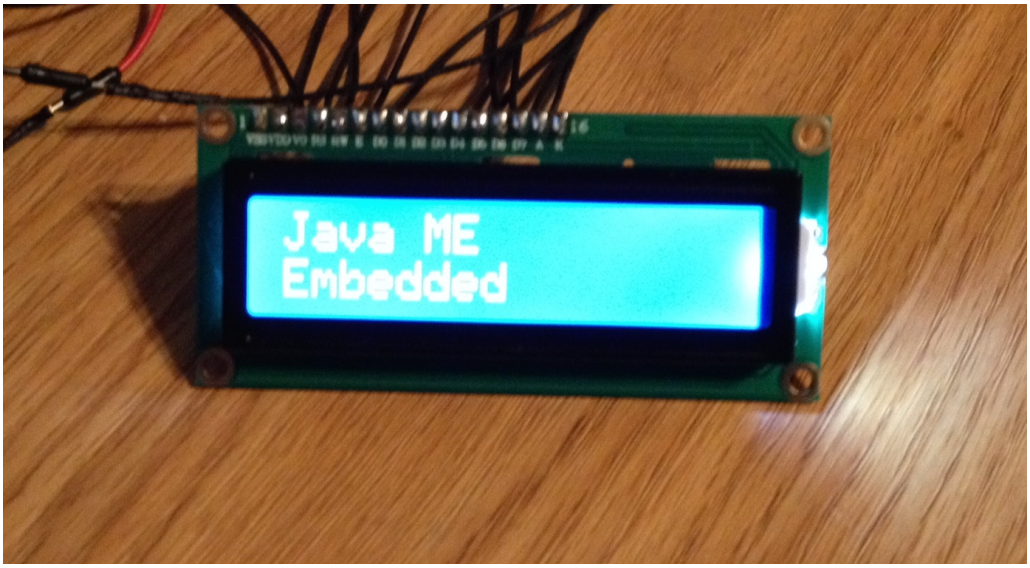
The following permissions must be added to the Application Descriptor of the project so that it will execute without any security exceptions from the Oracle Java ME Embedded runtime.

Table 6-10 API Permissions for LCD Example

Permission	Device	Operation
`jdk.dio.DeviceMgmtPermission`	`*:*`	`open`
`jdk.dio.i2cbus.I2CPinPermission`	`*:*`	`open`

After running the application, you should see the display as shown in Figure 6-6.

Figure 6-6 LCD Display after Running Example



7

The Serial Peripheral Interface (SPI) Bus

The Serial Peripheral Interface or SPI bus is a synchronous serial data link that operates in full duplex mode. In other words, data can be sent and received at the same time. Devices communicate in master/slave mode, where the master device initiates the data exchange with one or more slaves. Multiple slave devices are allowed with individual *slave select* lines.

The SPI bus specifies four logic signals:

- SCLK : Serial Clock (a clock signal that is sent from the master).
- MOSI : Master Output, Slave Input (data sent from the master to the slave).
- MISO : Master Input, Slave Output (data sent from the slave to the master).
- SS : Slave Select (sent from the master, active on low signal). Often paired with the Chip Select (CS) line on an integrated circuit that supports SPI.

In order to enable the SPI bus on the Raspberry Pi, uncomment the entry `spi_bcm2708` in the file `/etc/modprobe.d/raspi-blacklist.conf`. Note that you will need to have root privileges to edit the file.

Using the SPI Bus to Communicate with an ADC

Because the Raspberry Pi board does not come with a analog-to-digital converter, the SPI bus can be used to communicate with a peripheral analog-to-digital converter chip that is reading an analog signal.

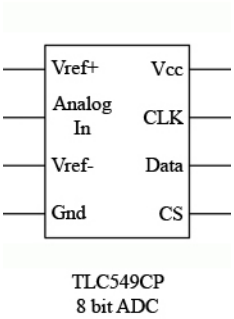
For this exercise, you will need the following hardware:

Table 7-1 Hardware for Example 3-1

Hardware	Where to Obtain
Raspberry Pi 512 MB Rev B	Various third-party sellers
Texas Instruments TLC549CP 8-bit ADC	Various electronics suppliers. We used mouser.com.
T-Cobbler and Breadboard	Adafuit. See Chapter 1 for more information.
Potentiometer	Electronics store
Jumper Wires (M/M and F/F)	Electronics store.

The data sheet of the TLC549CP shows 8 pins, as shown in Figure 7-1. Note that the SPI connections reside on the right side of the chip, while the connections for measuring the analog signal are on the left side of the chip.

Figure 7-1 Pinouts for TLC549CP Analog-to-Digital Converter Chip



In order to connect the TLC549CP chip to the Raspberry Pi, the SPI connections must be connected as shown in Table 7-2.

Table 7-2 Raspberry Pi to TLC549CP SPI Pins

Pins on Raspberry Pi	TLC549CP ADC Board Pins (Right Side)
3.3V	VCC
SCLK (GPIO 11 / Pin 23)	CLK
MISO (GPIO 9/ Pin 21)	Data
CE0 (GPIO 8 / Pin 24)	CS

The other four pins must be connected to provide the analog voltage to measure. In this example, we are using a potentiometer (in effect, a variable resistor) to vary the amount of voltage being sent into the Analog In pin.

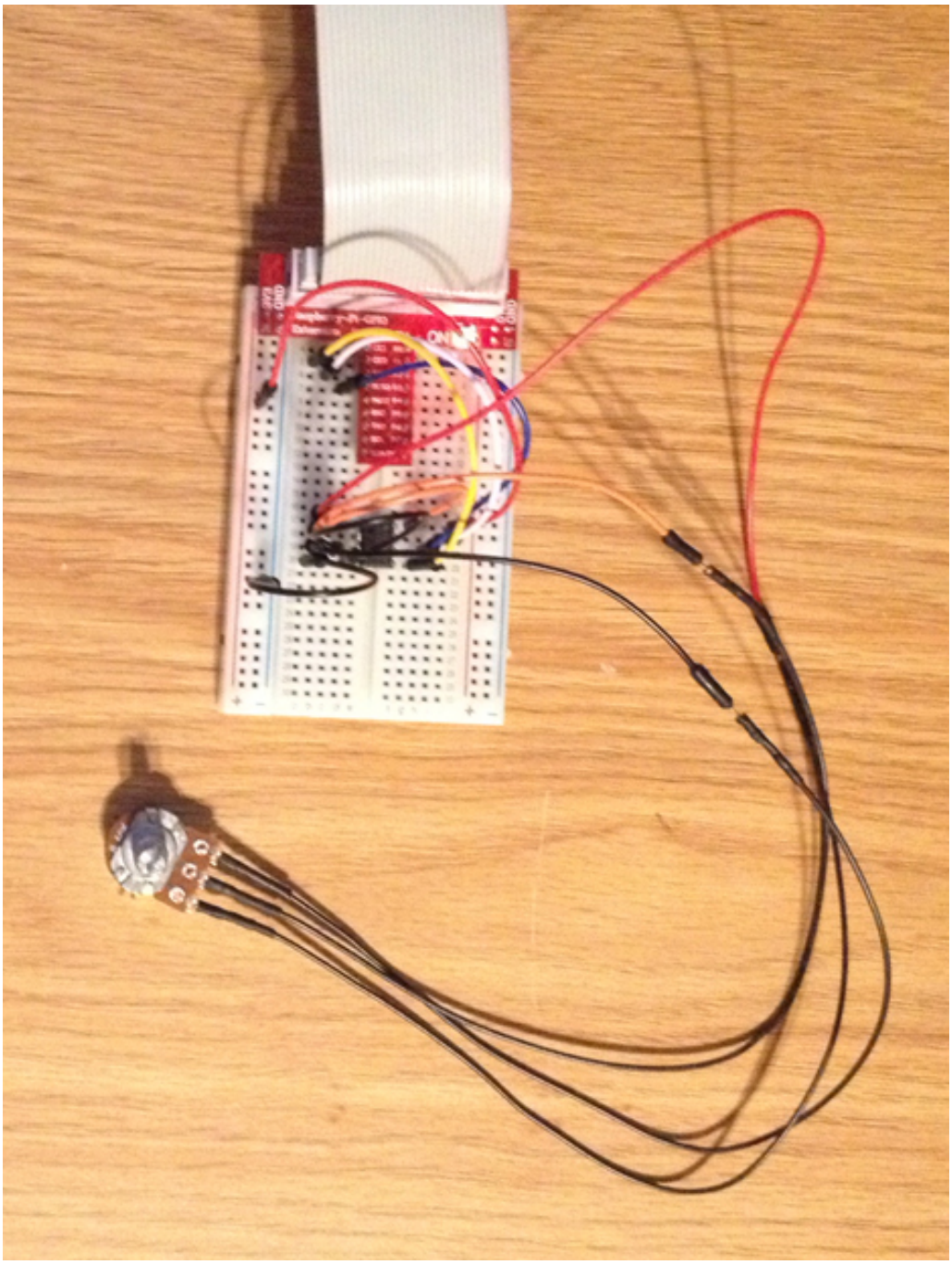
Table 7-3 shows how to connect the remaining pins on the TCL549CP chip.

Table 7-3 TLC549CP to Analog Signal Pins

TLC549CP ADC Board Pins (Left Side)	Analog Signal
Vref+	Voltage (Side Pin on Potentiometer) / 3.3V
Analog In	Variable Voltage Signal (Middle Pin on Potentiometer)
Vref-	Voltage (Other Side Pin on Potentiometer)
GND	To Ground

Note that in order to complete our circuit and provide power to the potentiometer, the Vref+ must be also connected to a 3.3V input, and the Vref- must be connected to a ground. The chip does not provide voltage. You can test the voltage that is being sent through the potentiometer with a voltmeter to ensure that the circuit is working properly. The completed circuit on the breadboard is shown in Figure 7-2.

Figure 7-2 Breadboard with the Analog-to-Digital Converter Circuit



Once this is completed, we can use the source code in the following example to test out the ADC chip.

```
import jdk.dio.Device;
import jdk.dio.DeviceManager;
import jdk.dio.spibus.SPIDevice;
import jdk.dio.spibus.SPIDeviceConfig;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.util.logging.Level;
import java.util.logging.Logger;
```

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


public class SPIExample1 extends MIDlet {

    public void startApp() {

        System.out.println("Preparing to open SPI device...");

        SPIDeviceConfig config = new SPIDeviceConfig(0, 0,
                SPIDeviceConfig.CS_ACTIVE_LOW,
                500000,
                3,
                8,
                Peripheral.BIG_ENDIAN);

        try (SPIDevice slave = (SPIDevice)DeviceManager.open(config)) {

            System.out.println("SPI device opened.");

            for (int i = 1; i < 200; i++) {
                ByteBuffer sndBuf = ByteBuffer.wrap(new byte[]{0x00});
                ByteBuffer rcvBuf = ByteBuffer.wrap(new byte[1]);
                slave.writeAndRead(sndBuf,rcvBuf);
                System.out.println("Analog to digital conversion at " +
                    i + " is: " + rcvBuf.get(0));
                Thread.sleep(1000);

            }

        } catch (IOException ioe) {
            // handle exception
        } catch (InterruptedException ex) {
            Logger.getLogger(SPIExample1.class.getName()).
                log(Level.SEVERE, null, ex);
        }

    }

    public void pauseApp() {
    }

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

This program is very simple: it opens up a connection to the Raspbeery Pi SPI bus using a `SPIDeviceConfig` and writes a byte to the peripheral device: the ADC chip. Since there is no input connection being sent from the master (the Raspberry Pi) to the slave (the ADC chip), this data is effectively ignored. The SPI bus will, concurrently, attempt to retrieve a byte of data from the chip. This byte is passed along the MISO line, which returns an 8-bit number that represents the current voltage level. This process will be repeated 200 times, with a one-second delay between each sampling on the bus.

The program output looks like the following. As the program is running, try turning the dial on the potentiometer to vary the voltage that is being sent into the chip. Here, we are turning the voltage from higher to lower, and the ADC chip is representing this with a steady drop in the 8-bit value that is returned.

```
Starting emulator in execution mode
...
About the open device
Device opened...
Value for 1 is: 145
Value for 2 is: 143
Value for 3 is: 120
Value for 4 is: 113
Value for 5 is: 90
Value for 6 is: 75
Value for 7 is: 63
```

8

Working with Java ME Encryption

Learn about the the encryption functionality available to the Java ME Embedded programmer with the Oracle Java ME Embedded 8.3 release.

Topics:

Connecting to an SSL Server

Creating a connection to an SSL server only requires the programmer to include an appropriate `ConnectionOption` object in the call to `Connector.open()`. This example requires the following hardware:

Table 8-1 Hardware for Example 1-1

Hardware	Where to Obtain
Raspberry Pi 512 MB Rev B, B+, or Raspberry Pi 2	Various third-party sellers

In this example, we use the Oracle Java ME Embedded runtime to connect to a server on the network that is running TLSv1.1 or higher on port 443. Note that this example requires the user to configure a web server that will accept an incoming connection on that port and uses the proper protocol and is properly signed by a valid certificate authority. After this is setup, the value of the `sTestServerAddr` variable should be changed accordingly. The following example shows the source code.

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import javax.microedition.io.ConnectionOption;
import javax.microedition.io.Connector;
import javax.microedition.io.SecureConnection;
import javax.microedition.midlet.MIDlet;


public class SSLConnect extends MIDlet {

    @Override
```

```
    public void startApp() {

        SecureConnection sc;
        ConnectionOption<String> protocol;
        InputStream is;
        OutputStream os;
        DataInputStream dis;
        DataOutputStream dos;

        String sTestServerAddr = "example.com:443";

        try {

            protocol = new ConnectionOption<>("Protocol", "TLSv1.1");
            sc = (SecureConnection) Connector.open("ssl://" +
                sTestServerAddr, protocol);

            System.out.println("Connection successful to:");
            System.out.println("Address: " + sc.getAddress());
            System.out.println("Port: " + sc.getPort());
            System.out.println("Cipher Suite: " +
                sc.getSecurityInfo().getCipherSuite());
            System.out.println("Protocol Name: " +
                sc.getSecurityInfo().getProtocolName());
            System.out.println("Protocol Version: " +
                sc.getSecurityInfo().getProtocolVersion());

        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    @Override
    public void pauseApp() {
    }

    @Override
    public void destroyApp(boolean unconditional) {
    }

}
```

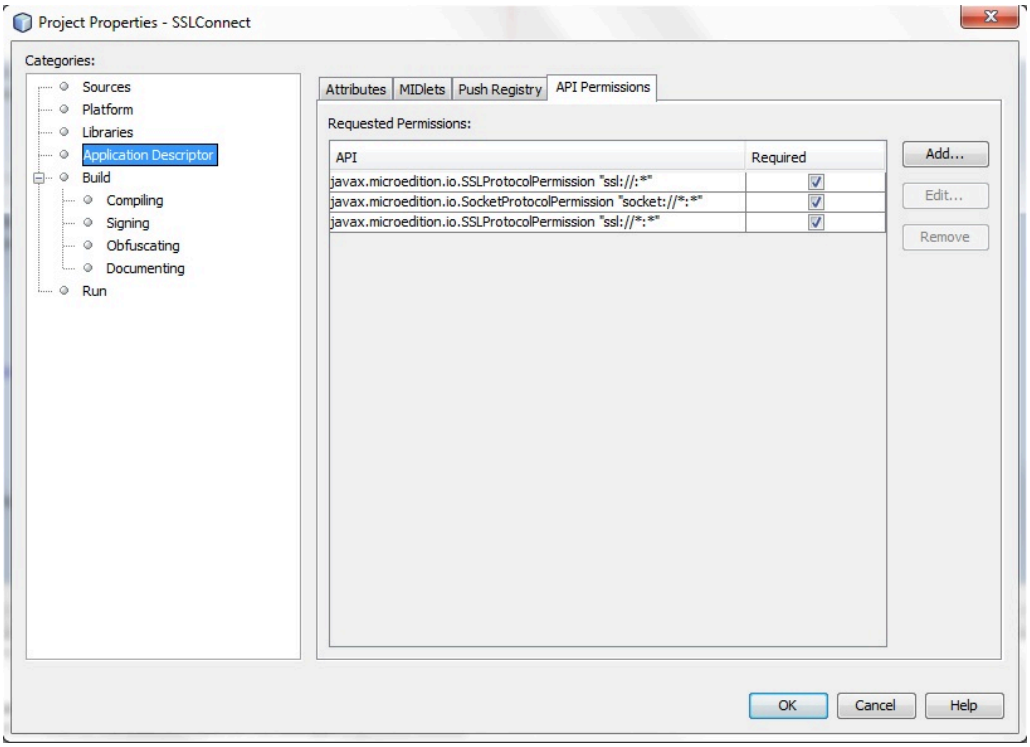
The following permissions must be added to the Application Descriptor of the IMlet so that it will execute without any security exceptions from the Oracle Java ME Embedded runtime.

Table 8-2 Permissions for Example 4-1

Permission	Device
`javax.microedition.io.SSLProtocolPermission`	`ssl://*:*`
`javax.microedition.io.SocketProtocolPermission`	`socket://*:*`

Note that if you're using an IDE such as NetBeans as the development environment, you will need to access the project properties of the project and set API permissions under the application descriptor, as shown in Figure 8-1.

Figure 8-1 API Permissions in the Project Properties Dialog in NetBeans



Tip:

If your server does not currently use a certificate from a signed certificate authority (CA), you can import a server certificate to the Java ME Embedded device. Locate the MEKeytool executable in the bin directory of the Oracle Java ME Embedded SDK distribution, and enter the following command using a Windows command prompt.

```
C:\SDK\bin> mekeytool.exe -import -Xdevice:EmbeddedExternalDevice1 -keystore
myCert.crt
```

This command will connect to the keystore on the embedded device currently recognized by the Device Manager as "EmbeddedExternalDevice1" and install the certificate with the filename `myCert.crt`. Note that this certificate must be identical to the one residing on the server that is authenticating SSL/TLS connections, or the Java embedded runtime will throw a `javax.microedition.pki.CertificateException` when attempting a secure connection. See Appendix D for more information on using the MEKeyTool utility.

After running the application, you should see output that identifies a successful connection to the server at the address and port specified. The program will then output the address and port, as well as the security connection parameters that were used to make the connection.

```
Connection successful to:Address: 192.168.1.125Port: 443Cipher Suite:
TLS_RSA_WITH_AES_256_CBC_SHAProtocol Name: TLSProtocol Version: 3.2
```

Authenticating an SSL Server

In this example, we expand on the `ConnectionOption` objects to provide an option to authenticate an HTTPS server. As with the previous example, the value of the `serverAddr` variable should be modified to point to a properly configured server. The following example shows the source code.

Note:

Oracle Java ME Embedded 8.2 has removed support for SSLv3 due to a widely-publicized security vulnerability. However, the source code example is applicable to other forms of transport-layer security included with Oracle Java ME Embedded 8.3.

```
import java.io.DataInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.microedition.io.ConnectionOption;
import javax.microedition.io.Connector;
import javax.microedition.io.HttpConnection;
import javax.microedition.io.HttpsConnection;
import javax.microedition.midlet.MIDlet;
import javax.microedition.rms.RecordStore;

/**
public class AuthenticateServer extends MIDlet {

    public static final int PASSED = 1;
    public static final int FAILED = -1;

    @Override
    public void startApp() {

        String serverAddr = "https://example.com:443";

        HttpsConnection hc;
        ConnectionOption<String> auth;
        ConnectionOption<String> protocol;
        int response = HttpConnection.HTTP_NOT_FOUND;

        try {

            auth = new ConnectionOption<>("AuthenticateServer", "TRUE");
            protocol = new ConnectionOption<>("Protocol", "TLSv1.1");

            hc = (HttpsConnection) Connector.open(serverAddr,
                Connector.READ_WRITE, auth, protocol);
            response = sendReqAndgetResp(hc); //request GET

            if (response == PASSED) {
                System.out.println("Pass");
            } else {
                System.out.println("Failed");
```

```
            }

        } catch (IOException ex) {
            ex.printStackTrace();
        } catch (SecurityException ex) {
            ex.printStackTrace();
        } catch (RuntimeException ex) {
            ex.printStackTrace();
        }
    }

    @Override
    public void destroyApp(boolean unconditional) {
    }

    private int sendReqAndgetResp(HttpsConnection hc) {

        int resCode = -1;
        boolean pass = true;

        OutputStream os;

        try {
            ((HttpsConnection) hc).setRequestMethod(HttpConnection.GET);
            resCode = ((HttpsConnection) hc).getResponseCode();
            System.out.println("Response code is: " + resCode);

            if (resCode == HttpConnection.HTTP_OK) {
                return PASSED;
            } else {
                return FAILED;
            }

        } catch (IOException ex) {
            ex.printStackTrace();
            return FAILED;
        } finally {
            try {
                hc.close();
            } catch (IOException ex) {
            }
        }
    }
}
```

The following permissions must be added to the Application Descriptor of the IMlet. Note that because we are using HTTPS, we require the HTTPS protocol permission, even through the implementing protocol we requested for HTTPS (TLSv1.1) is the same.

Table 8-3 Permissions for Example 4-2

Permission	Device
`javax.microedition.io.HTTPSProtocolPermission`	`https://*:*`

This example is similar to the previous example. Here, however, we create an HTTPS connection with requests and responses (instead of a direct SSL connection). An

additional `ConnectionOption` object also instructs the Java ME example to authenticate the server.

```
auth = new ConnectionOption<>("AuthenticateServer", "TRUE");
```

Enabling this option will verify that the server certificate is valid and has been signed by a valid certificate authority, as well as performing a number of verification steps against the data presented by the certificate. If the test is successful, you should see output that identifies a connection to the HTTPS server at the address and port specified.

```
Pass
```

Accessing the Keystore

Each Java ME Embedded implementation has one or more keystores, typically located under the `appdb/certs` directory. There is one keystore for each application privilege level (such as `untrusted` or `operator`). In order to programmatically access the keystore, use the classes in the `com.oracle.crypto.keystore` package.

The following example shows source code used to create five certificates, store them in the keystore, and then iterate over the contents of the keystore when completed.

```
import java.io.DataInputStream;
import com.oracle.crypto.cert.X509Certificate;
import com.oracle.crypto.cert.X509CertificateBuilder;
import com.oracle.crypto.keypair.KeyPair;
import com.oracle.crypto.keypair.KeyPairGenerator;
import com.oracle.crypto.keypair.PrivateKey;
import com.oracle.crypto.keypair.spec.RSAKeyGenParameterSpec;
import com.oracle.crypto.keystore.KeyStore;
import com.oracle.crypto.keystore.KeyStoreEntry;
import com.oracle.crypto.keystore.KeyStoreException;
import java.security.spec.AlgorithmParameterSpec;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.microedition.midlet.MIDlet;


public class GenerateKeystore extends MIDlet {

    @Override
    public void startApp() {

        try {

            HashMap<String, KeyStoreEntry> shadow = new HashMap();
            KeyStore ks = KeyStore.getInstance(KeyStore.STORAGE.CLIENT);

            for (int i = 0; i < 5; i++) {

                KeyStoreEntry kse = generateRandomKeyStoreEntry();
                System.out.println("Add entry with certificate serial number: " +
                        kse.getCertificate().getSerialNumber());
                ks.addEntry(kse);

            }
```

```
            List<KeyStoreEntry> list = ks.getEntries();
            Iterator<KeyStoreEntry> iter = list.iterator();

            while (iter.hasNext()) {
                KeyStoreEntry kse = iter.next();
                String subject = kse.getCertificate().getSubject();
                System.out.println("Certificate Subject: " + subject);

                PrivateKey entryKey = kse.getPrivateKey();
                byte[] entryEncoded = entryKey.getEncoded();

                System.out.println("Private Key: " + entryEncoded.toString());

            }

        } catch (SecurityException ex) {
            //  Handle exception
        } catch (KeyStoreException ex) {
            //  Handle exception
        }

    }

    @Override
    public void destroyApp(boolean unconditional) {
    }

    private KeyStoreEntry generateRandomKeyStoreEntry() {

        KeyStoreEntry entry = null;
        try {

            AlgorithmParameterSpec param;
            param = new RSAKeyGenParameterSpec(512, 3);

            KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
            kpg.initialize(param);
            KeyPair kp = kpg.generateKeyPair();

            int serialNumber = (int)(Math.random() * 1000000);

            System.out.println("Create X509 Certificate serial: " + serialNumber);

            X509CertificateBuilder builder = new X509CertificateBuilder(kp);
            builder.setSerialNumber("" + serialNumber);
            builder.setSubject("C=JP;ST=NE;L=Menlo;O=Oracle;OU=Java;CN=Test." +
                serialNumber);
            builder.setValidityInDays(365);
            X509Certificate cert = builder.create();

            entry = new KeyStoreEntry(cert, kp.getPrivate(), "test");

        } catch (Throwable ex) {
            ex.printStackTrace();
        }
        return entry;
    }
}
```

The following permissions must be added to the Application Descriptor of the IMlet to access the keystore on the Java ME Embedded device.

Table 8-4 Permissions for Example 4-3

Permission	Device
com.oracle.crypto.keystore.KeyStorePermission	client_only

This example will access the local keystore on the embedded board (client) with the following call:

```
KeyStore ks = KeyStore.getInstance(KeyStore.STORAGE.CLIENT);
```

Note that the keystore that is accessed will depend on the trust level of the application. If the Java ME embedded application is not signed, it will fall into the `untrusted` security domain by default.

We can access the keystore similar to accessing it with the Java SE environment. First, we create a `KeyStoreEntry` object and populate it with a certificate. This is, in turn, added to the embedded keystore via a simple loop and iterated over later in the program. Here is the output after running the program:

```
Creating X509 Certificate with serial number: 798364Add keystore entry with
certificate serial number: 4F:53:40Creating X509 Certificate with serial number:
67079Add keystore entry with certificate serial number: 43:07:09Creating X509
Certificate with serial number: 723418Add keystore entry with certificate serial
number: 48:22:12Creating X509 Certificate with serial number: 792956Add keystore
entry with certificate serial number: 4F:1D:38Creating X509 Certificate with serial
number: 661145Add keystore entry with certificate serial number: 42:0B:2D

Certificate Subject: C=JP,ST=NE,L=Menlo,O=Oracle,OU=Java,CN=Test.798364Private Key:
[B@fcd4cfc2Certificate Subject: C=JP,ST=NE,L=Menlo,O=Oracle,OU=Java,CN=Test.
67079Private Key: [B@1c1cc1a5Certificate Subject:
C=JP,ST=NE,L=Menlo,O=Oracle,OU=Java,CN=Test.723418Private Key:
[B@e854b14cCertificate Subject: C=JP,ST=NE,L=Menlo,O=Oracle,OU=Java,CN=Test.
792956Private Key: [B@1d69c567Certificate Subject:
C=JP,ST=NE,L=Menlo,O=Oracle,OU=Java,CN=Test.661145Private Key: [B@1a5796e6
```

Configuring the Board as a Secure Server

The Java ME Embedded binary contains functionality that enables an embedded board to function as a server using secure protocols. The functionality is identical to the configuration of a Java SE server.

The secure server connection requires a server certificate on the device. This certificate should be imported into the device with a private key.

```
mekeytool.exe -import -Xdevice:EmbeddedExternalDevice1 -keystore myServertCert.jks -
storepass <store password> -keypass <private key password>
```

The following example shows source code used to setup the embedded board as a server.

```
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.microedition.io.ConnectionOption;
import javax.microedition.io.Connector;
```

```
import javax.microedition.io.SecureConnection;
import javax.microedition.io.SecureServerConnection;
import javax.microedition.midlet.MIDlet;

public class EmbeddedServer extends MIDlet {

    @Override
    public void startApp() {

        String PORT = "10005";

        SecureServerConnection ssc = null;
        ConnectionOption<String> protocol = null;

        try {

            protocol = new ConnectionOption<>("Protocol", "TLSv1.1");
                 ConnectionOption serverCert = new ConnectionOption("Certificate",
<Certificate Subject DN>);
            ssc = (SecureServerConnection) Connector.open("ssl://:"+PORT,
                protocol, serverCert);

            System.out.println("Connection listening on:");
            System.out.println("Address: " + ssc.getLocalAddress());
            System.out.println("Port: " + ssc.getLocalPort());

            ssc.acceptAndOpen();

            System.out.println("Connection made!");

        } catch (SecurityException ex) {
            ex.printStackTrace();
        } catch (RuntimeException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

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

Note: In this code example the `<Certificate Subject DN>` parameter should be replaced with the Subject DN of the used certificate.

The following permissions must be added to the Application Descriptor of the IMlet to access the keystore on the Java ME Embedded device.

Table 8-5 Permissions for Example 4-3

Permission	Device
javax.microedition.io.SSLProtocolPermission	ssl://*:*
javax.microedition.io.SSLProtocolPermission	ssl://:*

This example uses a `SecureServerConnection` object, which creates a socket at port 10005 using the SSL protocol. It is essential to include the following line as well:

```
ssc.acceptAndOpen();
```

Without this line, the secure socket will never accept an incoming connection, and any attempt to make a connection will result in a runtime exception. Note that a call to this method will block until a successful connection is made.

As with the previous examples, the Java ME Embedded keystore must have a valid server certificate for the trust-level that the application is running under, which will be validated upon any secure connection. If the certificate is absent, or is not valid, an exception will be thrown on the server.

The client-side code to make the connection is nearly identical to the client-side example earlier in this chapter, with a slight change to the port and the protocol. A successful connection should output the following:

```
Address: 192.168.1.80
Port: 10005
Connection made!
```

A

Java ME Optimization Techniques

This appendix covers common optimization techniques when working with Java ME Embedded devices. Many of these techniques are common to the CLDC VM in both Java Embedded and traditional Java ME.

Design

Embedded systems are typically designed to perform a specific task, unlike a general-purpose computer that strives to handle multiple tasks with equal efficiency. Some embedded systems also have real-time performance constraints for safety and usability; others may have little or no performance requirements, allowing the system hardware to be simplified to reduce costs. As such, developers should use the simplest application design possible to avoid overtaxing the embedded system.

Memory

Resources are frequently limited in embedded devices. Often memory is the most valuable resource. Many embedded devices have memory that is measured in megabytes (MB), and some of it is used by the Runtime Operating System (RTOS), leaving the remainder for use by the Java VM and its applications.

Be aware of how much memory is typically used by your application, the RTOS, and the Java VM. This will vary from one embedded board to another. By the time a Java ME embedded application exhausts all memory and is subject to an `OutOfMemoryError`, there are few options left: the application must either force the VM to free any unnecessary memory using a `System.gc()` call, or if that doesn't work, crash.

Threads

Java threads are often an expensive resource with embedded Java VMs. Java embedded applications work best when using minimal application threads. If you must create multi-threaded code, be sure to minimize the use of synchronized code, which can be expensive on embedded devices. As a general rule, avoid using the `Timer` class, as an extra thread is created for each timer.

A common technique for creating Java ME embedded applications is to create a background thread in the `startApp()` method of the `MIDlet` class and reuse it throughout the IMlet lifecycle.

System Callbacks

System callback functions should never block and should return as soon as possible to avoid slowing down the CLDC VM. Pay special attention to the following `MIDlet` methods:

- `startApp()`

- `pauseApp()`
- `MIDlet` constructor

Input/Output

The Record Management System (RMS) is an I/O resource that should be used carefully. With any application that uses `RecordStore` objects, opening and closing operations should be minimized. In addition, strive to group reads and writes in one section of code as much as possible. Spreading `RecordStore` read and writes across the application can slow down the application.

Another common strategy when working with `RecordStore` objects is to use buffers, which reside in memory and are often faster. This is a common technique:

- For reading record stores, read the entire record into a buffer, then parse the buffer.
- For writing record stores, write to a single buffer, then write the buffer to a record.

General Tips

Here are some other general hints for optimizing your code that are pervasive throughout the industry for Java ME code:

- Object creation is very costly with respect to memory and performance overhead. Create objects only when needed, and reuse any object instances that are created in a cache.
- Use lazy instantiation if appropriate. However, many Java ME developers will create all objects outside the main loop of the program and reuse them as the application runs. With reusable objects, be sure to include a method that returns them to the original state, independent of the object constructor.
- Avoid auto-boxing when possible.
- Do not perform assertions in tight loops.
- Avoid using variable-length arguments (*varargs*).
- Use local variables instead of global variables when you can. Local variables are faster and generate less bytecode.
- Only include system classes that you need. Avoid using wild character imports like `import java.util.*` Instead, import classes directly, such as `import java.util.Date`.
- Don't perform string concatenations using the "+" operator. Use the `StringBuffer` class instead. For example, don't do the following:

  ```
  String str = new String ("Hello  ");str += "World";
  ```

  Instead, do this:

  ```
  StringBuffer str = new StringBuffer ("Hello ");str.append("World");
  ```

  Remember that in Java, `String` objects are immutable, so performing concatenation with the "+" operator will in fact create a `StringBuffer`, copy the contents of the `String` over, append the other `String`, then copy the result back into a different immutable `String` object.

- Divide your multi-dimensional arrays into single-dimensional arrays. Multidimensional arrays take more time to calculate the proper index in memory.
- Avoid any unnecessary creation and disposal of objects and variables inside loops. For example, avoid a construct like this:

```
for (int i = 0; i < length; i++) {  MyConstantClass c = new MyConstantClass();
results[i] = c.doSomething();}
```

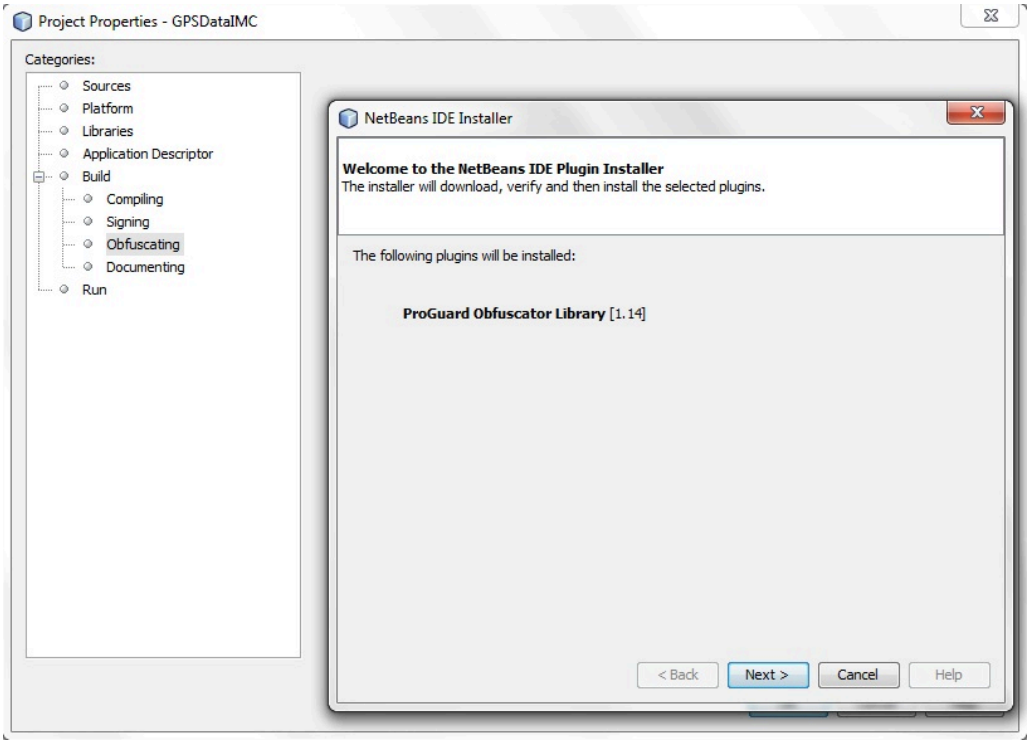
- Use a `switch-case` construct instead of `if` blocks, as they are compiled into more optimized bytecode. Remember that starting with Java 8, the `switch` keyword can handle strings, which is more efficient that creating a large number of `if` blocks that test using the `equals()` method.
- Use public variables directly instead of using get/set accessors.
- Set variables to `null` when you don't need them anymore to assist with garbage collection.

Application Size

Aside from minimizing the number of classes in your application, developers can also make use of obfuscator tools, which are present in NetBeans and other IDEss. The original purpose of an obfuscator is to make reverse engineering bytecode more difficult. However, it can also create smaller and often faster class files. In fact, obfuscators typically reduce Java ME embedded class file size by 25% to 35%.

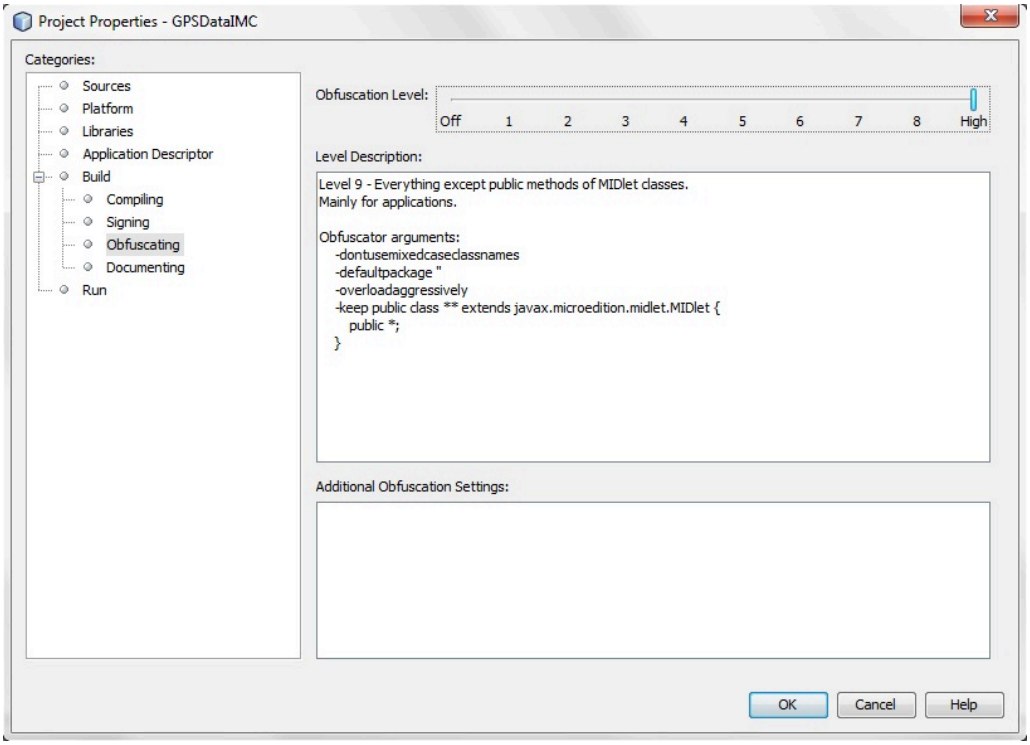
The NetBeans IDE contains an option to install the ProGuard obfuscator library. You can choose this option by right-clicking on your project and bringing up the **Project Properties**. Next, expand the **Build** leaf and select **Obfuscating**. If ProGaurd is not already installed, press the button to download and install the NetBeans module, as shown in Figure A-1.

Figure A-1 Installing the ProGuard Obfuscator Library



Once the obfuscator is installed, choose an obfuscation level by moving the slider anywhere from Level 1 and Level 9. As shown in Figure A-2, each level presents a detailed description in the window below that shows what operations the obfuscator is performing.

Figure A-2 Choosing an Obfuscation Level



B

Java ME Embedded Properties

This appendix documents the configurable options that are found in many ports of the Oracle Java ME Embedded product. System properties in the Oracle Java ME Embedded distribution can be configured in one of two ways: by modifying the `jwc_properties.ini` file (if available), or by using the VM proxy command-line interface (CLI).

Modifying the Properties File

Most platforms have a `jwc_properties.ini` file that can be modified with a text editor. The `jwc_properties.ini` file has two distinct sections:

- `[application]`

  Properties that are used by Java applications that are running on the board.

- `[internal]`

  Properties used for internal system configuration.

In addition, each `jwc_properties.ini` file contains comments that help describe the purpose of each entry.

It is highly recommend that you read through the *original* `jwc_properties.ini` file for your target embedded board, as it contains essential information about each property. Note that the Oracle Java ME Embedded runtime may alter the values inside of the `jwc_properties.ini` file at any time (especially if the `set-property` and `save-properties` commands are issued in the CLI), and typically without comments, so it helps to study the original version that comes with each distribution bundle.

Using the Command-Line Interface

In addition to specifying properties in the `jwc_properties.ini` file, system properties can also be modified on the fly using the CLI `set-property` command. The `set-property` command uses the following syntax:

`set-property` <*key*> <*value*>

If you wish to examine the current value of any property, use the `get-property` command.

`get-property` <*key*>

Note that after any property change, the VM state is unpredictable, and it is necessary to issue the `save-properties` command and '`shutdown -r`' (or, depending on the embedded board, cycle the power) to activate the changes. See Chapter 2 for more information on using the VM proxy CLI.

The list of configurable system properties differs extensively on each platform and with each release, and may be retrieved by issuing the following command via the CLI proxy:

```
> properties-list
```

For example, the properties list that is generated for the Raspberry Pi would look similar to the following:

```
read only       STRING  xml.rpc.subset.version = 1.0
read/write      STRING  xml.jaxp.subset.version = 1.0
read/write      BOOL    vmconfig.system_reboot = false
read only       STRING  system.storage_root = ../appdb
read/write      BOOL    system.network.reconnect = false
read/write      INT     system.jam_space = 4096000
read only       STRING  system.default_storage = ../appdb
read only       STRING  socket = com.sun.midp.io.j2me.socket.ProtocolPushImpl
read/write      STRING  security.providers.jar = null
...
(several lines omitted)
...
read/write      INT     AMS_MEMORY_LIMIT_MVM = -1
```

C

Signing an IMlet Suite's JAR File

Establishing trust is important for IMlet suites that use security-sensitive APIs. Signing an IMlet suite's JAR file allows the suite to be trusted. A JAR file is signed with the `jadtool` utility. A copy of the `jadtool` utility is provided with the Oracle Java ME Embedded software bundle.

The `jadtool` utility signs a JAR file by adding a certificate and the JAR file's digital signature to a Java Application Descriptor (JAD) file. Adding a certificate and a JAR file's digital signature to a JAD file are separate steps. You must complete both steps to sign a JAR file. The steps are in "Instructions for Using JadTool".

You can also use the `jadtool` utility to obtain information about a certificate in a JAD file. The information can include the name of the entity that issued the certificate, the certificate's serial number, the dates between which it is valid, and its Message Digest Algorithm 5 (MD5) and Secure Hash Algorithm (SHA) fingerprints.

Instructions for Using JadTool

This section explains how to use the `JadTool` utility through an example that signs a hypothetical IMlet suite named ImaginaryIMlet.

Note:

ImaginaryIMlet is not an actual IMlet suite. No ImaginaryIMlet files are included with this release.

The example uses the key pair provided with the software. The key pair is in the `j2se_test_keystore.bin` file, which is a keystore managed with the Java SE platform's `keytool` utility. For information on the `keytool` utility, see `http://download.oracle.com/javase/8/docs/technotes/tools/windows/keytool.html`.

After you build an implementation of the software, `j2se_test_keystore.bin` is located in this directory:

{OUTPUT-Dir}`/meep/bin/i386`

Where *{OUTPUT-Dir}* is the directory that contains the output of your builds of the Oracle Java ME Embedded software.

The password for the file is `keystorepwd`. The alias of the key pair is `dummyca.` The private key password is `keypwd`. The file is provided for testing purposes.

For IMlet suites on end-user devices, use an RSA key pair backed by a certificate or certificate chain from a certificate authority. You must import the certificate or certificate chain into a Java SE platform's keystore with the Java SE platform's `keytool` utility.

The `JadTool` utility is packaged in a JAR file named `JadTool.jar` in this directory where *{OJMEE-Dir}* is the base directory of the Oracle Java ME Embedded installation:

{OJMEE-Dir}`\toolkit-lib\process\jadtool\code`

Using the JadTool Utility

1. Open a Windows command prompt.
2. Change your current directory to the directory that holds your IMlet's JAR and JAD files.
3. Add the certificate for your key pair to the JAD file using the `JadTool` utility.

   The `JadTool` utility adds the certificate as the value of an attribute named `MIDlet-Certificate-`*m*`-`*n*, where *m* is the number of the certificate chain (it defaults to one but you can provide a different number with the `-chainnum` switch), and *n* is an integer that, for new certificates, begins at one and increments by one each time you add a new certificate to the JAD file.

   For example, if *{OUTPUT-Dir}*`/binaries` is `C:\jme\src\output\binaries`, the following command adds the certificate as the value of the attribute `MIDlet-Certificate-1-1` to the example JAD file:

   ```
   C:\myIMlets>java -jar %OJMEE_HOME%\toolkit-lib\process
   \jadtool\code\JadTool.jar -addcert -alias dummyca -storepass
   keystorepwd -keystore C:\jme\src\output\midp\bin
   \i386\j2se_test_keystore.bin -inputjad ImaginaryIMlet.jad -
   outputjad ImaginaryIMlet.jad
   ```

4. (Optional) Verify that the certificate is added to the JAD file by using the `JadTool` utility to list the certificate in the JAD file.

   ```
   C:\myIMlets>java -jar %OJMEE_HOME%\toolkit-lib\process
   \jadtool\code\JadTool.jar -showcert -certnum 1 -inputjad
   ImaginaryIMlet.jad
   ```

   ```
   Subject: C=US, ST=CA, L=Santa Clara, O=dummy CA, OU=JCT, CN=thehost
   Issuer : C=US, ST=CA, L=Santa Clara, O=dummy CA, OU=JCT, CN=thehost
   Serial number: 3d3ece8a
   Valid from Wed Jul 24 08:58:02 PDT 2002 to Sat Jul 21 08:58:02 PDT 2012
   Certificate fingerprints:
     MD5: 87:7f:5e:64:c8:dd:b4:bf:35:39:76:87:99:9b:68:82
     SHA: 9d:c0:88:ce:08:83:cd:e6:fe:13:8b:26:f6:b4:df:e2:da:3c:25:98
   ```

5. If you have a key pair backed by a certificate chain, import the intermediate certificates.

   Import the intermediate certificates using the `JadTool` utility with the `-addcert` switch shown in Step 3, taking care to use the correct chain order.

   For example:

   The *XXXX* company provides a certificate that vouches for your key pair, the WidgetCertificates company vouches for the *XXXX* certificate, and VeriSign vouches for the WidgetCertificates certificate.

   Import the *XXXX* certificate followed by the WidgetCertificate. The *XXXX* certificate is `MIDlet-Certificate-1-2` and the WidgetCertificate certificate is `MIDlet-Certificate-1-3`.

Note:

You do not import the certificate of the root CA. In this example, the certificate is from VeriSign. The root certificate is on the device.

6. Sign the JAR file using the `JadTool` utility.

   The `JadTool` utility signs the JAR file, base64 encodes the signature, and stores it as the value of the `MIDlet-Jar-RSA-SHA1` attribute of the output JAD file.

   Note:

   The key used to sign the JAR file must be from the same Java SE keystore entry as key pair specified in Step 3. The `JadTool` utility does not check that the JAR file is signed with a keystore entry that has a certificate in the JAD file.

   For example:

```
C:\myIMlets>java -jar %OJMEE_HOME%\toolkit-lib\process
\jadtool\code\JadTool.jar -addjarsig -keystore C:\jme\src
\output\midp\bin\i386\j2se_test_keystore.bin -alias dummyca -
storepass keystorepwd -keypass keypwd -jarfile
ImaginaryIMlet.jar -chainnum 2 -inputjad ImaginaryIMlet.jad -
outputjad ImaginaryIMlet.jad
```

   Optionally, you can sign a JAR file with a stronger signature (JAD attribute `MIDlet-Jar-RSA-SHA256`) using the `-useSha256` command line switch.

   For example, `java -jar jadtool.jar -addjarsig -useSha256 -keystore j2se_test_keystore.bin -alias dummyca -storepass keystorepwd -keypass keypwd -jarfile ImaginaryIMlet.jar -inputjad ImaginaryIMlet.jad -outputjad ImaginarySignedSha256IMlet.jad`

Handling Expired Certificates

A JAD file can have multiple certificate, but it can hold the signature for only one JAR file. When a certificate in the JAD file expires, you must add a new certificate and re-sign the JAR file. When re-signing the JAR file, the `JadTool` utility overwrites the current digital signature with the new one.

Options Summary

The `jadtool` utility supports the following options:

- **none**

  Running the tool without options returns the same information as the `-help` option.

- `-addcert -alias` *keyAlias* `[ -keystore` *keystore* `] [ -storepass` *password* `] [ -chainnum` *chainNumber* `] [ -certnum` *certNumber* `] [ -encoding` *encoding* `] -inputjad` *inputJadFile* `-outputjad` *outputJadFile*

Adds a certificate to a JAD file. To add a certificate, this utility first creates the certificate from the entry identified by *keyAlias* in keystore. The keystore, if provided, must be a Java Cryptography Architecture keystore (a file containing data such as key

entries in a format that the Java SE platform can use). If `keystore` is not provided, its default, *{User_Home_Dir}*`/.keystore`, is used. If keystore requires a password to access its contents, password must be provided.

After creating the certificate and attribute name, this utility concatenates the contents of *inputJadFile* with the new certificate and writes it as *outputJadFile*.

You can use the same file for the *inputJadFile* and *outputJadFile*.

The certificate is in the JAD file as the value of an attribute named MIDlet-`Certificate-`*m*`-`*n*, where:

- *m* is *chainNumber*, or 1 if it is not provided. A JAD file can contain multiple certificate chain.
- *n* is *certNumber*. The value *certNumber* depends on whether the new certificate replaces an existing certificate. If the certificate is a replacement, then *certNumber* must be the number of the certificate to replace. For example, if the new certificate would replace the one stored as the value of attribute `MIDlet-Certificate-1-3`, then *certNumber* must be `3`. If the certificate is new, *certNumber* is ignored.

  If *inputJadFile* uses an encoding other than UTF-8 (ASCII with unicode escapes), *encoding* must be specified. This utility uses the same encoding for reading *inputJadFile* and writing *outputJadFile*.

- `-addjarsig [-useSha256][ -jarfile` *jarFile* `] -alias` *keyAlias* `[ -keystore` *keystore* `] -storepass` *storePassword* `-keypass` *keyPassword* `[ -encoding` *encoding* `] [-chainnum` *chainNumber* `] -inputjad` *inputJadFile* `-outputjad` *outputJadFile*

  Creates a digital signature for *jarFile*. If *jarFile* is not specified, the value of the `MIDlet-Jar-URL` attribute from *inputJadFile* is used. The attribute's value must be a valid HTTP URL.

  This utility creates a digital signature for the JAR file using the private key identified by *keyAlias* in *keystore*. If *keystore* is not provided, its default is `{User_Home_Dir}/.keystore`. This utility gets the key from *keystore* using *storePassword* and *keyPassword*, and creates the signature with it using the EMSA-PKCS1-v1_5 encoding method of PKCS #1, version 2.2. See RFC 2437 at `http://www.ietf.org/rfc/rfc2437.txt`.

  After creating the signature, this utility concatenates the contents of *inputJadFile* with the signature, and writes it as *outputJadFile*. The signature is base64 encoded, and is in the output JAD file as the value of the `MIDlet-Jar-RSA-SHA1-`*m* attribute where m is chainNumber, or 1 if it is not provided. This number corresponds to the value for *m* in the `MIDlet-Certificate-`*m*`-`*n* attribute.

  If *inputJadFile* uses an encoding other than UTF-8 (ASCII with unicode escapes), *encoding* must be specified. This utility uses the same encoding for reading *inputJadFile* and writing *outputJadFile*.

- `-help`

  Prints a usage summary.

- `-showcert [( [-certnum` *certNumber* `] [ -chainnum` *chainNumber* `] ) | -all ] [ -encoding` *encoding* `] -inputjad` *inputJadFile*

Prints information about either all certificates, or the certificate that corresponds to the given *certNumber* and *chainNumber* in the *inputJadFile*. The option `-all` cannot be combined with the `-certnum` and `-chainnum` options.

The *chainNumber* of a certificate is the *m* in the JAD file's `MIDlet-Certificate-`*m*`-`*n* attribute, while the *certNumber* is the *n*. For example, to show the certificate that is the value of attribute `MIDlet-Certificate-2-3`, the *chainNumber* must be `2` and *certNumber* must be `3`. If *certNumber* or *chainNumber* are not provided (and the `-all` option is not used), the utility uses a 1.

The information printed includes the certificate's subject, issuer, serial number, dates between which it is valid, and fingerprints (md5 and SHA). The attributes in the subject and issuer names are shown in reverse order from what is in the certificate (a side effect of using the Java SE platform certificate API). As a result, the names might not match what is returned from other tools that display a certificate's subject and issuer names.

If *inputJadFile* uses an encoding other than UTF-8 (ASCII with unicode escapes), *encoding* must be specified. The tool uses the same encoding for reading *inputJadFile* and writing *outputJadFile*.

D

Managing Keys and Certificates

The Oracle Java ME Embedded platform uses public keys from a Certificate Authority (CA) to validate Web sites and signed IMlet suites. The Oracle Java ME Embedded implementation also uses private keys from certificates to establish secure connections with client authentication. When the platform uses a secure protocol to access a Web site, the site provides a certificate which is typically signed by a CA. In the same manner, signed IMlet suites also contain a certificate that is signed by a CA. The Oracle Java ME Embedded platform checks the validity of a certificate by using the CA's public key. By signing a certificate, a CA certifies the identify of the owner of the Web site or IMlet suite.

You can manage the CA certificates, public keys, and private keys on the embedded board by using the `MEKeyTool` utility. The `MEKeyTool` utility is provided with the Java ME SDK distribution, and is similar to the `keytool` utility provided with the Java SE platform, except that `MEKeyTool` will also operate across a network on the keystores of an embedded device that is currently recognized by the Oracle Java ME Embedded Device Manager.

This chapter describes how to use `MEKeyTool` to manage keystores that are used by the Oracle Java ME Embedded Emulator or recognized by the Oracle Java ME SDK Device Manager.

Running MEKeyTool

`MEKeyTool` is an executable that can be found in the following location:

{ME-SDK_Home-Dir}`\bin\MEKeyTool.exe`

Where *ME-SDK-Home-Dir* is the base directory of the Oracle Java ME Embedded installation.

To connect to a keystore on an embedded board instead of the emulator, add the `-Xdevice` option to reference a device currently recognized by the Oracle Java ME Embedded Device Manager:

```
MEKeyTool.exe -Xdevice:EmbeddedExternalDevice1
```

Warning:

Each embedded board may only accept a limited subset of commands, depending on the functionality offered. To list the functionality supported by the device `EmbeddedExternalDevice1`, for example, you can use the emulator's Xquery option:

```
emulator.exe -Xquery -Xdevice:EmbeddedExternalDevice1
```

Using the MEKeyTool Utility

1. Open a command prompt or terminal window.
2. Change your current directory to the location of the EXE file shown above, or add the directory to your current %PATH%.
3. Run `MEKeyTool` with the options needed.

   For example, use the following command to display help:

   ```
   MEKeyTool.exe -help
   ```

ME Keystores

The `MEKeyTool` utility keeps the CA certificates, public keys, and private keys in an ME *keystore*. Depending on the device, the keystore is at the following locations, where *base-dir* is the base directory of the Oracle Java ME Embedded or Oracle Java ME SDK installation.

Table D-1 Location of Keystores

Device	Location
Emulator	{*base_dir*}`/runtimes/meep/appdb/certs`
Embedded Boards	*[base_dir*}`/appdb/certs` (*if file system is accessible*)

This keystore directory contains an index file named `_main.ks` and a set of certificate files. The platform includes the key of one CA.

Warning:

Oracle does not recommend modifying the default keystore, but instead modifying a copy, either one that is user-generated or in the working directory for the appropriate device.

Working Directory for the Emulator

When the Oracle Java ME Embedded emulator is first started, it creates a working directory in {*User_Home_Dir*}`/javame_sdk/`*{Version}*`/work/`*{device}*`/appdb/certs`. (Note that this is only the case for emulated devices, such as `EmbeddedDevice1`, not devices that map to actual embedded boards such as `EmbeddedExternalDevice1`.) Next, it copies all certificates and several other important files there. Be aware that deleting the working directory removes all device settings and, of course, any additions to the local keystore.

Note:

The `MEKeytool` utility enables you to import keys from Java SE keystores. However, you cannot use the `MEKeytool` utility directly on a Java SE keystore. For example, if you try to use the `MEKeytool` utility to view public keys in a Java SE keystore, the utility displays an error message that the keystore is corrupted. An ME keystore has a different format Java SE platform

keystores, which have a format in accordance with the Java Cryptography Architecture specification.

Creating and Managing Multiple ME Keystores

In addition to managing the public keys in the ME keystore, you can use the `MEKeytool` utility to manage additional ME keystores, both with the emulator and some embedded boards that contain an accessible filesystem. For example, during testing you might want to have multiple keystores to run against. One keystore could contain all the necessary testing keys, a second keystore could contain a subset of the testing keys, and a third keystore could contain an expired key.

Creating Alternate ME Keystores

The `MEKeytool` utility does not create a new ME keystore directory. The developer must create an empty keystore first, consisting of a directory and an empty `_main.ks` file, which can be copied from the keystore provided with the distribution bundle. See Importing a Key for instructions on how to import a key.

Managing Alternate ME Keystores

To manage a keystore other than the default, use the `-import -MEkeystore` option:

`MEKeyTool.exe -import -MEkeystore` *keystoreName ...*

For example, if you created an ME keystore, *{User_Home_Dir}*`\myKeys\set2_test_keys.ks`, that contains the keys that are needed to run a particular set of tests, use the `MEKeyTool` command to manage that keystore:

`MEKeyTool.exe -import -MEkeystore` *{User_Home_Dir}*`/myKeys/set2_test_keys.ks ...`

For all `MEKeyTool` commands, you receive an error message if the file that you provide as an argument to `-MEkeystore` does not exist.

Importing a Key

You can add a key to an ME keystore by importing it from the Java Cryptography Architecture keystore that comes with the Java SE platform or from a keystore that you create. For more information on the keystore that comes with the Java SE platform, see `http://download.oracle.com/javase/8/docs/technotes/tools/windows/keytool.html`.

The file name for the Java SE keystore is `.keystore` and the default location is in your home directory. This file is created if you use the Java SE platform's `keytool` utility to create keys and you do not specify a different location. The `MEKeyTool` utility references this keystore unless you use the `-keystore` argument to specify a different keystore.

Note:

If you use a Java SE keystore other than the default, the new keystore might require a password.

The `-import` option imports a key. For example, to add a key with an alias `dummyca` from the Java SE keystore `j2se_test_keystore.bin` that has a password,

`keystorepwd`, to the ME keystore at *{User_Home_Dir}*/`myKeys/set2_test_keys.ks`:

```
C:\>MEKeyTool.exe -import -alias dummyca -keystore j2se_test_keystore.bin -
storepass keystorepwd -MEkeystore myKeys/set2_test_keys.ks
```

If it is necessary to import a certificate with a private key from a file in PKCS12 format (for example, `cert_with_key.p12`) with the keystore password `storepwd` and the key password `keypwd`, use the command:

```
C:\>MEKeyTool.exe -import -keystore cert_with_key.p12 -storepass storepwd -
keypass keypwd -MEkeystore myKeys\set2_test_keys.ks
```

Listing Available Keys

An ME keystore organizes the keys that it contains by giving each one a number. For each key, the keystore also holds the name of the entity to whom the public key belongs, the time over which the key is valid, and the domain associated with the key. The `MEKeyTool -list` command displays information for each key in a particular keystore.

The following example lists the contents of the ME keystore *{User_Home_Dir}*/`myKeys/set1_test_keys.ks`:

```
C:\>MEKeyTool.exe -list -MEkeystore myKeys/set1_test_keys.ks
Key 1
  Owner: C=US;O=VeriSign, Inc.;OU=Class 3 Public Primary Certification Authority
  Valid from Mon Jan 29 03:00:00 MSK 1996 to Wed Aug 02 03:59:59 MSD 2028
  Security Domain: identified_third_party
  Enabled: true
Key 2
  Owner: O=Oracle;C=myserver
  Valid from Sat Aug 03 00:43:51 PDT 2002 to Tue Jul 31 00:43:51 PDT 2012
  Security Domain: operator
  Enabled: true
```

The following example lists of the contents of the ME keystore on a Raspberry Pi device currently listed as `EmbeddedExternalDevice1`. Again, note that the device must already be registered with the device manager.

```
C:>MEKeyTool.exe -list -Xdevice:EmbeddedExternalDevice1
[1]
Owner CN=AddTrust External CA Root,OU=AddTrust External TTP Network,O=AddTrust
AB,C=SE valid from Tue May 30 03:48:38 PDT 2000 till Sat May 30 03:48:38 PDT 2020

[2]
Owner CN=GlobalSign Root CA,OU=Root CA,O=GlobalSign nv-sa,C=BE valid from Tue Sep
01 05:00:00 PDT 1998 till Fri Jan 28 04:00:00 PST 2028

[3]
Owner CN=GTE CyberTrust Global Root,OU=GTE CyberTrust Solutions\, Inc.,O=GTE
Corporation,C=US valid from Wed Aug 12 17:29:00 PDT 1998 till Mon Aug 13 16:59:00
PDT 2018

[4]
Owner CN=Entrust.net Secure Server Certification Authority,OU=(c) 1999 Entrust.net
Limited,OU=www.entrust.net/CPS incorp. by ref. (limits liab.),O=Entrust.net,C=US
valid from Tue May 25 09:09:40 PDT 1999 till Sat May 25 09:39:40 PDT 2019

[5]
```

```
Owner OU=Class 3 Public Primary Certification Authority,O=VeriSign\, Inc.,C=US
valid from Sun Jan 28 16:00:00 PST 1996 till Tue Aug 01 16:59:59 PDT 2028

[6]
Owner OU=VeriSign Trust Network,OU=(c) 1998 VeriSign\, Inc. - For authorized use
only,OU=Class 3 Public Primary Certification Authority - G2,O=VeriSign\, Inc.,C=US
valid from Sun May 17 17:00:00 PDT 1998 till Tue Aug 01 16:59:59 PDT 2028

[7]
Owner CN=thehost,OU=Unknown,O=TEST,L=Unknown,ST=Unknown,C=US valid from Wed Nov 16
11:40:27 PST 2005 till Sat Nov 14 11:40:27 PST 2015

[8]
Owner CN=GeoTrust CA for UTI,O=Unified Testing Initiative (UTI),C=US valid from
Thu Jan 22 21:00:00 PST 2004 till Tue Jan 23 20:55:00 PST 2024

[9]
Owner
1.2.840.113549.1.9.1=#16197072656d69756d2d736572766572407468617774652e636f6d,
CN=Thawte Premium Server CA,OU=Certification Services Division,O=Thawte Consulting
cc,L=Cape Town,ST=Western Cape,C=ZA valid from Wed Jul 31 17:00:00 PDT 1996 till
Thu Dec 31 15:59:59 PST 2020

[10]
Owner OU=Equifax Secure Certificate Authority,O=Equifax,C=US valid from Sat Aug 22
09:41:51 PDT 1998 till Wed Aug 22 09:41:51 PDT 2018
```

Deleting a Key

When keys expire, you must delete them from the keystore and add their replacements. You can also delete unused keys. For example, if you added the public key of a test site with a self-signed certificate during testing, you can delete that key when testing is completed.

The `-delete` command to the `MEKeyTool` utility removes a key from an ME keystore. The `-delete` command requires one of the following options:

- `-owner` *ownerName*

  Sets the string that describes the owner of the public key in a given keystore. Use the `-list` command to print information about each key in the keystore. The string in the command must match the one printed when you use the `-list` command to the `MEKeyTool` utility. See Listing Available Keys for more information.

- `-number` *keyNumber*

  Sets the number that a given keystore has assigned to each of its keys. The number is greater than or equal to one. Use the `-list` command to print the number that the keystore has assigned to each of its keys. See Listing Available Keys for more information.

The following examples show two ways to delete a key from the ME keystore *{User_Home_Dir}*`/myKeys/set1_test_keys.ks` (the keystore used in Listing Available Keys):

- Deleting a key by using its key number-

  ```
  C:\>MEKeyTool.exe -delete -number 1 -MEkeystore myKeys
  \set1_test_keys.ks
  ```

- Deleting a key by using its owner name-

```
C:\>MEKeyTool.exe -delete -owner "C=US;O=VeriSign,\
Inc.;OU=Class 3 Public Primary Certification Authority" -
\MEkeystore myKeys\set1_test_keys.ks
```

Replacing a Key

Some situations require that you replace a key (such as when a key expires). To replace a key, first delete the old key, then import the new key.

Note:

If you import the new key before deleting the old one, the `MEKeyTool` utility displays an error message that the owner of the key has a key in the ME keystore.

MEKeyTool Summary

The `MEKeyTool` utility supports the following options:

- *no option*

  Runs the tool without options and returns the same information as the `-help` option.

- `-help`

  Prints a usage summary.

- `-import` [`-alias` *keyAlias*] [`-keystore` *JavaSEKeystore*][`-keypass` *keyPassword*][`-storepass` *storePassword*][`-client` *clientName*]

  Imports a key identified by security client *clientName* or *keyAlias* from *JavaSEKeystore* into the device keystore. If *JavaSEKeystore* is not provided, its default, *{User_Home_Dir}*`/.keystore`, is used (where *{User_Home_Dir}* is the user's home directory).

  If *JavaSEKeystore* requires a password, you must provide *storePassword*. If the `-keypass` argument is provided, the private key will be imported to the ME keystore together with public certificate.

- `-list` [`-client` *clientName*]

  Lists the number, owner, and validity period, and domain of the key identified by security client *clientName*, or all keys if the option is omitted, in the device keystore.

- `-delete` [`-client` *clientName*] (`-owner` *ownerName* | `-number` *keyNumber*)

  Deletes the key identified by security client *clientName*, *ownerName* or *keyNumber* from the device keystore.

  You can provide either *ownerName* or *keyNumber*, but not both. You can find the valid values for them by running the `MEKeyTool` utility with the `-list` command.

- `-export` [`-client` *clientName*] (`-number` *keyNumber*) (`-out` *outputFile*)

  Exports a certificate, specified by security client *clientName* or *keyNumber*, from the device keystore to the output file *outputFile*. The format of the *outputFile* is:

  - PEM in the case of an extracted public key or certificate
  - PKCS#12 in the case of an extracted certificate with a private key. The keystore password of the PKCS#12 file is the same password that was used in the `-keypass` parameter of the import command

- `-clients`

  Presents a list of all the security clients defined in the system that can accept public keys.

E

OEM Extensions

This chapter describes the OEM Extensions, which provide a mechanism to add extensions to the binary runtime of the Oracle Java ME Embedded software.

Using OEM Extensions

The Oracle Java ME Embedded software enables you to extend the binary runtime by making your own Java packages available to IMlets as OEM extensions. You specify the location of the JAR files that contain your packages in a configuration file and IMlets can use those packages at run time.

To make your packages available to IMlets as OEM Extensions, follow these steps:

1. Write the Java classes in your package.
2. Compile your classes.
3. Preverify your classes.
4. Create a JAR file that contains your classes.
5. Add the JAR file to the configuration file `jwc_properties.ini`.
   a. Edit the file `jwc_properties.ini`. This file is located in `runtimes/meep/bin/jwc_properties.ini` in the Windows distribution and in `java/jwc_properties.ini` in the distribution for the reference board.
   b. Locate the line that contains the configuration setting `extraclasspath`.
   c. Add the location of your JAR file. Use a semicolon between paths if you have more than one JAR file, for example:

      ```
      extraclasspath = C:/myjar1.jar;C:/myjar2.jar
      ```

      Note:

      You must use forward slashes in the paths in `extraclasspath`.
6. Share the JAR file and the details of your packages with the IMlet programmers that need to use your classes.

F

Encryption Algorithms

The Java ME 8 product includes the following supported cipher suites and encryption algorithms, with specified key lengths.

Supported Algorithms for Windows, Linux, and Raspberry Pi Platforms

TLSv1.0-1.2 are supported with the following configurations.

Note:

The following cipher suites are disabled by default but can be enabled in the `jwc_properties.ini` file by modifying the `SSL_FORBIDDEN_CIPHERS_FILTER` property:

1) `TLS_ECDH_anon_*` : these are non-secure anonymous cipher suites. In order to enable these, replace "`:!aNULL`" with "`:aNULL`" in the properties file.

2) `*_WITH_NULL_*` : these are non-secure unencrypted cipher suites. In order to enable these, replace "`:!eNULL`" with "`:eNULL`" in the properties file.

TLSv1.0 - TLSv1.2

Refer to the list of cipher suites in the priority ordering.

```
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_AES_256_GCM_SHA384
TLS_RSA_WITH_AES_256_CBC_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_PSK_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256
```

```
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_128_CBC_SHA
SSL_RSA_WITH_IDEA_CBC_SHA
TLS_PSK_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA
TLS_PSK_WITH_3DES_EDE_CBC_SHA
SSL_RSA_WITH_DES_CBC_SHA
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
```

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 suite that contains attributes used by application management software (AMS) to manage the MIDlet's life cycle, and other application-specific attributes used by the MIDlet 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 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.