Oracle® Java ME Embedded

Developer's Guide

Release 8

E52611-01

April 2014

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 Build Guide, Release 8

E52611-01

Contents

List of Examples

List of Figures

List of Tables

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.

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

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

Operating System Commands

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

Shell Prompts

Shell	Prompt
Bourne shell	$

Shell	Prompt
Windows	*directory>*

Conventions

The following text conventions are used in this document:

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

1

Developer Migration Guide

This chapter discusses the changes between version 3.4 of the Oracle Java ME Embedded and the current instance, version 8. It is designed as a guide to help application developers port earlier applications to the latest version of the Oracle Java ME Embedded runtime. If you have not developed IMlets using version 3.4 or earlier of the Oracle Java ME Embedded platform, you can safely skip this chapter.

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

In a typical profiling and monitoring session, the Java virtual machine must do a large amount of extra work: collecting, storing, and analyzing data, as well as replying to requests from external tools. When this is done on embedded devices, possibly using a slower CPU or constrained memory, development can become an unacceptably sluggish experience.

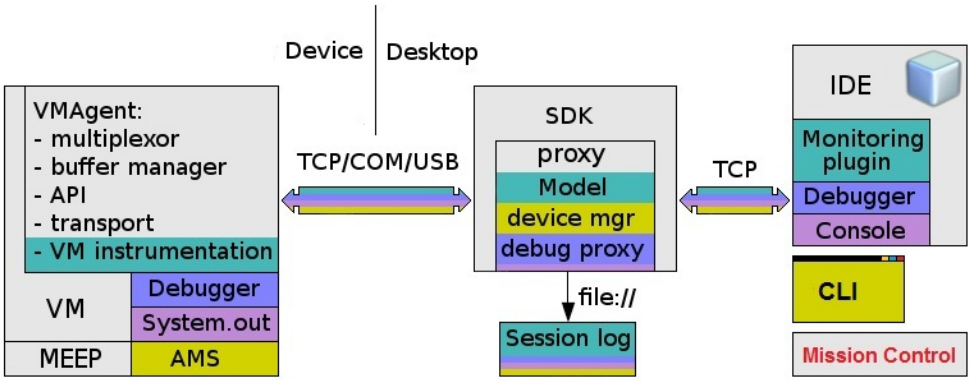
For this reason, version 8 of the Oracle Java ME Embedded software moves as much CPU intensive processing away from the embedded Java VM as possible. Instead, a separate application running on the host side will interact across the network with the internals of the Java VM. With this design, the VM only 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 and the VM proxy channel becomes the inter-component tool, and Javacall, CLDC, MEEP, JSRs and SDK components can all take advantage of it.

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. The current supported platforms are: Win32 (the emulator), RPi (Raspberry Pi with an embedded Linux OS), Keil (RTX OS), Orion (Brew MP OS), and STMicro Discovery.

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
```

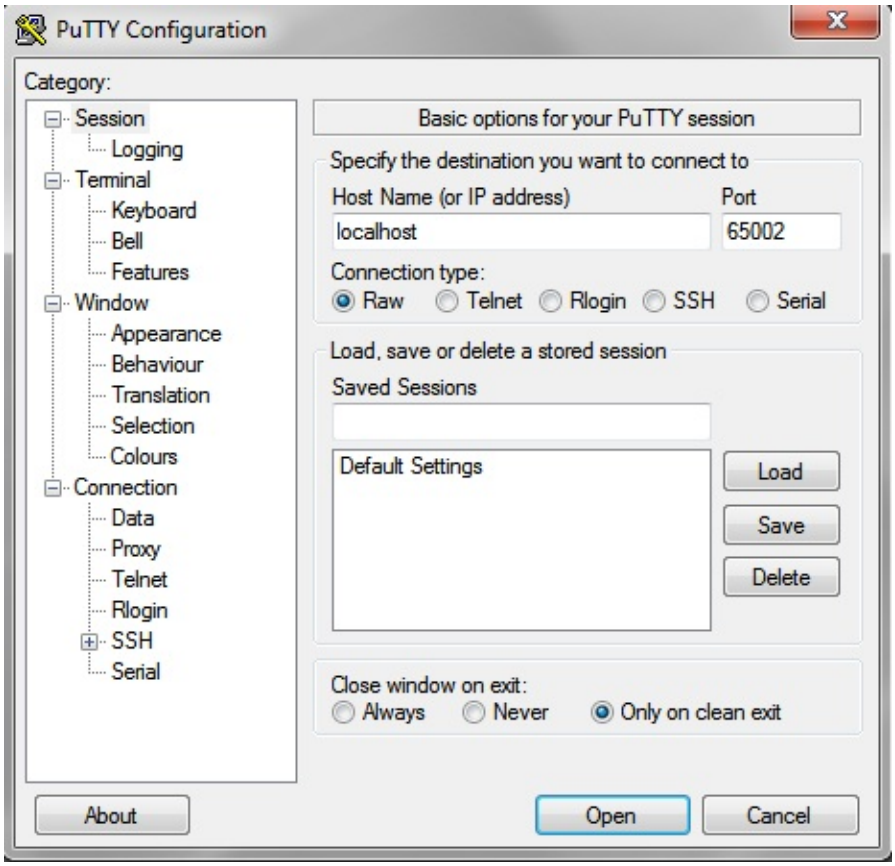
-serial <COM_PORT> - runs the proxy with a serial transport. This means that the VM proxy communicates with device across the specified serial port.

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

Using the Command Line Interface

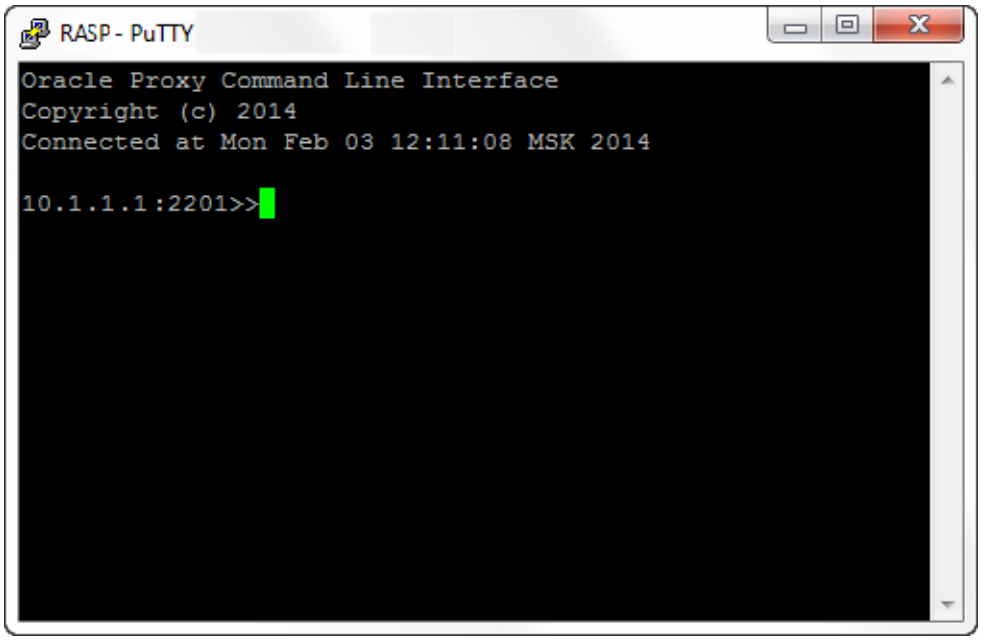
Once the VM proxy is running on the desktop, you can use the AMS CLI. The easiest way to do this is to start a PuTTY executable on your desktop computer, and connect to `localhost` at port 65002. This is shown in Figure 2–1. See the appropriate Getting Started Guide for your embedded board for platform-specific 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: unexpected. see help.	An unexpected parameter was found.
<<ams-install,ERROR,duplicate parameter: auth. see help.	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.
<*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

Response	Description
<<ams-list,0.name | vendor,status	List of installed suites with details
...	
<<ams-list,N.name | vendor,status	
<<ams-list,OK,N suites are installed	
<<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	List of the installed midlets in the suite. Note that each suite status can be RUNNING or STOPPED.
...	
<<ams-list,N.midlet,status	
<<ams-list,OK,N midlets are installed in suiteName | suiteVendor	

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
<<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.

Response	Description
<<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.
<<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

Response	Description
<<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.
<<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
`-i`	Displays additional property information

Responses

This command may return the following responses:

Response	Description
<<get-property,OK,imc = com.sun.midp.io.j2me.imc.ProtocolPushImpl	The property was found (without displaying additional information)
<<get-property,OK,dummy.property =	The property value is empty or not a set (without additional information)
<<get-property,OK, read/write STRING imc = com.sun.midp.io.j2me.imc.ProtocolPushImpl	The property is found (with `-i` flag)

Response	Description
<<get-property,OK, read/write STRING dummy-property =	The property value is empty or not a set (with `-i` flag)
<<get-property,FAIL,invalid parameters	An unexpected parameter was found
<<get-property,Usage: get-property name [-i]	
<<get-property,shows value of string property 'name'	
<<get-property, -i display property info	
<<get-property,FAIL,illegal argument [info]	The wrong flag format was used(e.g. using `-info` instead of `-i`)
<<get-property,Usage: get-property name [-i]	
<<get-property,shows value of string property 'name'	
<<get-property, -i display property info	
<<get-property,FAIL,connection is closed	An `IOException` has occurred.

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,Usage: set-property name value	
<<set-property,sets 'value' to property 'name'	

Response	Description
<<set-property,FAIL,illegal argument [microedition.devicevendor.isunique] or [true].	The property is read-only:
<<set-property,Usage: set-property name value	
<<set-property,sets 'value' to property 'name'	
<<set-property,FAIL,invalid parameters.	Wrong number of parameters:
<<set-property,Usage: set-property name value	
<<set-property,sets 'value' to property 'name'	
<<set-property,FAIL,connection is closed	An `IOException` has occurred

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.
<<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. This command only works on Qualcomm IoE devices.

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
<<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.
<<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
<<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

Response	Description
<<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

pwd

Prints the current working directory on the device.

Usage

`pwd`

Responses

This command may return the following responses:

Response	Description
<<pwd,OK	The command processed successfully
c:\Users\abc\javame-sdk\8.0_ea\work\EmbeddedDevice1\appdb	
<<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

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.

Permissions for Accessing Peripherals

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 Proposal for Java ME 8* specification and the associated Javadocs at the following site:

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

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
`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 the generic classes
`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 MMIO capabilities on the board
`jdk.dio.PeripheralMgmtPermission`	Use of any peripherals on the board

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

Permission	Description
`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
`javax.microedition.apdu.APDUPermission`	Use of an APDU device (e.g., card reader) on a board
`javax.microedition.cellular.CellularPermission`	Use of cellular telephone functionality on a board.
`javax.microedition.event.EventPermission`	Use of events
`javax.microedition.io.CommProtocolPermission`	Use of a communications 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 (DLTS) 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
`javax.microedition.midlet.ActionsDeniedPermission`	A permission to deny actions on a device
`javax.microedition.midlet.AutoStartPermission`	A permission to autostart an IMlet suite on a device
`javax.microedition.pim.ContactList.read`	Read a contact list
`javax.microedition.pim.ContactList.write`	Write to a contact list

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

Permission	Description
`javax.microedition.pim.EventList.read`	Read from an event list (calendar)
`javax.microedition.pim.EventList.write`	Write to an event list (calendar)
`javax.microedition.pim.ToDoList.read`	Read a to-do list
`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

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 1.0* specification and the associated Javadocs at the following site:

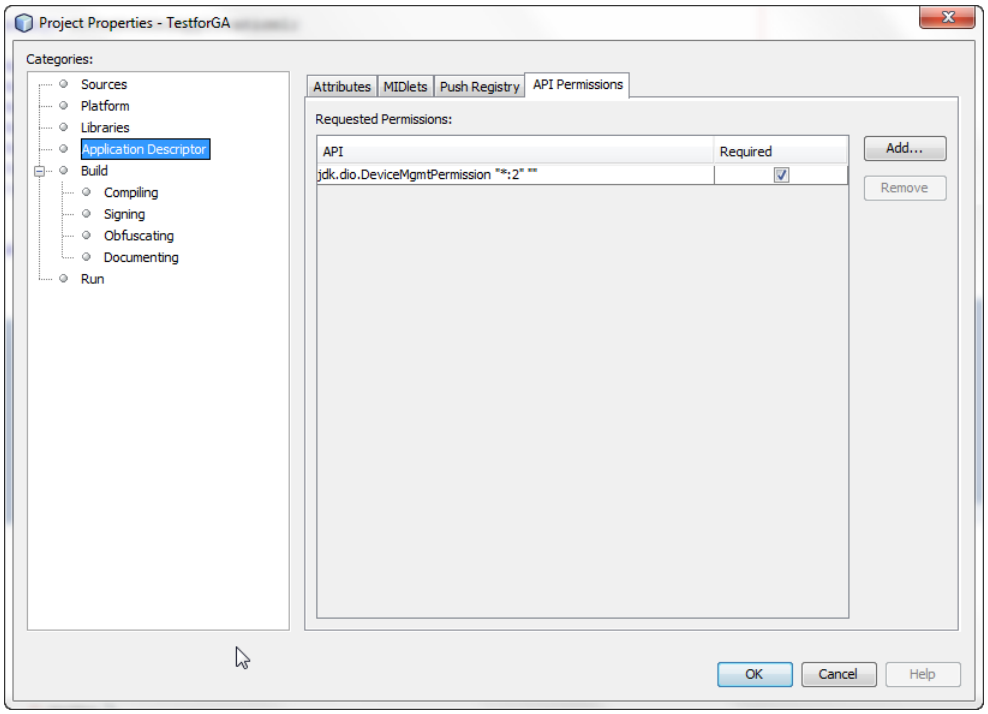
`http://docs.oracle.com/javame/embedded/embedded.html`

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

This method is more complex, but 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 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.

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
```

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

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.

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" on page 4-8 for more information on 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 `SuiteInstaller` interface is a sub-interface that consists of only two methods: one that starts the installation and one that cancels the installation.

- `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 hidden, and should not be visible to the user.
`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.

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

Table 4–4 (Cont.) Installer Error Codes

Constant	Error Code	Description
CANCELED	101	Canceled by user.
CANNOT_AUTH	35	The server does not support basic authentication.
CIRCULAR_COMPONENT_DEPENDENCY	64	Circular dynamic component dependency.
COMPONENT_DEPS_LIMIT_EXCEEDED	65	Dynamic component dependencies limit exceeded.
COMPONENT_NAMESPACE_COLLISION	72	The namespace used by a component is the same as another.
CONTENT_HANDLER_CONFLICT	55	The installation of a content handler would conflict with an already installed handler.
CORRUPT_DEPENDENCY_HASH	71	A dependency has a corrupt hash code.
CORRUPT_JAR	36	An entry could not be read from the JAR.
CORRUPT_PROVIDER_CERT	5	The content provider certificate cannot be decoded.
CORRUPT_SIGNATURE	8	The JAR signature cannot be decoded.
DEVICE_INCOMPATIBLE	40	The device does not support either the configuration or profile in the JAD.
DUPLICATED_KEY	88	Duplicated JAD/manifest key attribute
EXPIRED_CA_KEY	12	The certificate authority's public key has expired.
EXPIRED_PROVIDER_CERT	11	The content provider certificate has expired.
INCORRECT_FONT_LOADING	82	A font that is contained with the JAR cannot be loaded.
INSUFFICIENT_STORAGE	30	Not enough storage for this suite to be installed.
INVALID_CONTENT_HANDLER	54	The `MicroEdition-Handler-`*<n>* JAD attribute has invalid values.
INVALID_JAD_TYPE	37	The server did not have a resource with the correct type or the JAD downloaded has the wrong media type.
INVALID_JAD_URL	43	The JAD URL is invalid.
INVALID_JAR_TYPE	38	The server did not have a resource with the correct type or the JAR downloaded has the wrong media type.
INVALID_JAR_URL	44	The JAR URL is invalid.
INVALID_KEY	28	A key for an attribute is not formatted correctly.
INVALID_NATIVE_LIBRARY	85	A native library contained within the JAR cannot be loaded.
INVALID_PACKAGING	87	A dependency cannot be satisfied.

Table 4–4 (Cont.) Installer Error Codes

Constant	Error Code	Description
INVALID_PAYMENT_INFO	58	Indicates that the payment information provided with the IMlet suite is incomplete or incorrect.
INVALID_PROVIDER_CERT	7	The signature of the content provider certificate is invalid.
INVALID_RMS_DATA_TYPE	76	The server did not have a resource with the correct type or the JAD downloaded has the wrong media type.
INVALID_RMS_DATA_URL	73	The RMS data file URL is invalid.
INVALID_SERVICE_EXPORT	86	A LIBlet that exports a service with a LIBlet Services attribute does not contain the matching service provider configuration information.
INVALID_SIGNATURE	9	The signature of the JAR is invalid.
INVALID_VALUE	29	A value for an attribute is not formatted correctly.
INVALID_VERSION	16	The format of the version is invalid.
IO_ERROR	102	A low-level hardware error has occurred.
JAD_MOVED	34	The JAD URL for an installed suite is different than the original JAD URL.
JAD_NOT_FOUND	2	The JAD was not found.
JAD_SERVER_NOT_FOUND	1	The server for the JAD was not found.
JAR_CLASSES_VERIFICATION_FAILED	56	Not all classes within JAR package can be successfully verified with class verifier.
JAR_IS_LOCKED	100	Component or MIDlet or IMlet suite is locked by the system.
JAR_NOT_FOUND	20	The JAR was not found at the URL given in the JAD.
JAR_SERVER_NOT_FOUND	19	The server for the JAR was not found at the URL given in the JAD.
JAR_SIZE_MISMATCH	31	The JAR downloaded was not the same size as given in the JAD.
MISSING_CONFIGURATION	41	The configuration is missing from the manifest.
MISSING_DEPENDENCY_HASH	67	A dependency hash code is missing.
MISSING_DEPENDENCY_JAD_URL	66	A dependency JAD URL is missing.
MISSING_JAR_SIZE	21	The JAR size is missing.
MISSING_JAR_URL	18	The URL for the JAR is missing.
MISSING_PROFILE	42	The profile is missing from the manifest.
MISSING_PROVIDER_CERT	4	The content provider certificate is missing.
MISSING_SUITE_NAME	13	The name of MIDlet or IMlet suite is missing.

Table 4–4 (Cont.) Installer Error Codes

Constant	Error Code	Description
MISSING_VENDOR	14	The vendor is missing.
MISSING_VERSION	15	The version is missing.
NEW_VERSION	32	This suite is newer that the one currently installed.
NO_ERROR	0	No error.
NOT_YET_VALID_PROVIDER_CERT	89	A certificate is not yet valid.
NOT_YET_VALID_CA_KEY	90	A CA's public key is not yet valid.
OLD_VERSION	17	This suite is older that the one currently installed.
OTHER_ERROR	103	Other errors.
PROXY_AUTH	51	Indicates that the user must first authenticate with the proxy.
PUSH_CLASS_FAILURE	48	The class in a push attribute is not in MIDlet-<*n*> attribute.
PUSH_DUP_FAILURE	45	The connection in a push entry is already taken.
PUSH_FORMAT_FAILURE	46	The format of a push attribute has an invalid format.
PUSH_PROTO_FAILURE	47	The connection in a push attribute is not supported.
REVOKED_CERT	62	The certificate has been revoked.
RMS_DATA_DECRYPT_PASSWORD	83	Indicates that a password is required to decrypt RMS data.
RMS_DATA_ENCRYPT_PASSWORD	84	Indicates that a password is required to encrypt RMS data.
RMS_DATA_NOT_FOUND	75	The RMS data file was not found at the specified URL.
RMS_DATA_SERVER_NOT_FOUND	74	The server for the RMS data file was not found at the specified URL.
RMS_INITIALIZATION_FAILURE	81	Failure to import RMS data.
SUITE_NAME_MISMATCH	25	The MIDlet or IMlet suite name does not match the one in the JAR manifest.
TOO_MANY_PROPS	53	Indicates that either the JAD or manifest has too many properties to fit into memory.
TRUSTED_OVERWRITE_FAILURE	52	Indicates that the user tried to overwrite a trusted suite with an untrusted suite during an update.
UNAUTHORIZED	33	Web server authentication failed or is required.
UNKNOWN_CA	6	The certificate authority (CA) that issued the content provider certificate is unknown.
UNKNOWN_CERT_STATUS	63	The certificate is unknown to OCSP server.

Table 4–4 (Cont.) Installer Error Codes

Constant	Error Code	Description
UNSUPPORTED_CERT	10	The content provider certificate has an unsupported version.
UNSUPPORTED_CHAR_ENCODING	61	Indicates that the character encoding specified in the MIME type is not supported.
UNSUPPORTED_PAYMENT_INFO	57	Indicates that the payment information provided with the MIDlet or IMlet suite is incompatible with the current implementation.
UNTRUSTED_PAYMENT_SUITE	59	Indicates that the MIDlet or IMlet suite has payment provisioning information but it is not trusted.
VENDOR_MISMATCH	27	The vendor does not match the one in the JAR manifest.
VERSION_MISMATCH	26	The version does not match the one in the JAR manifest.

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 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.

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. Example 5–1 shows the source code.

Example 5–1 Setting a GPIO Pin

```
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);
        }
```

```
    }
}
```

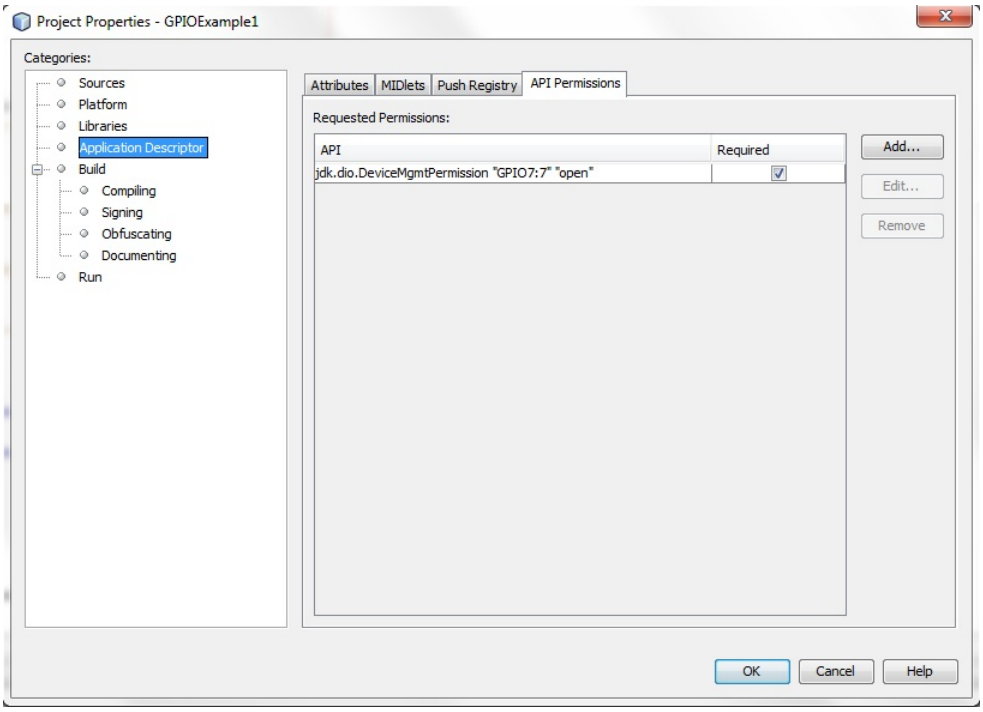
The following permissions must be added to the Application Descriptor of the IMlet so that it will execute without any security exceptions from the Oracle Java ME Embedded runtime.

Table 5–2 Permissions for Example 1-1

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

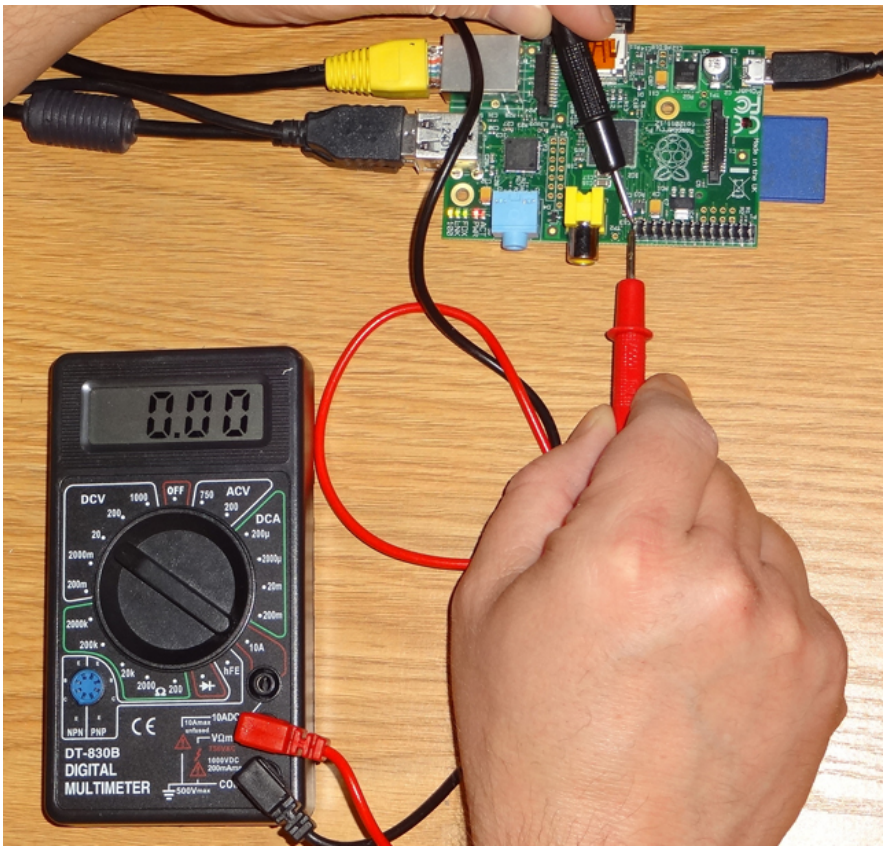
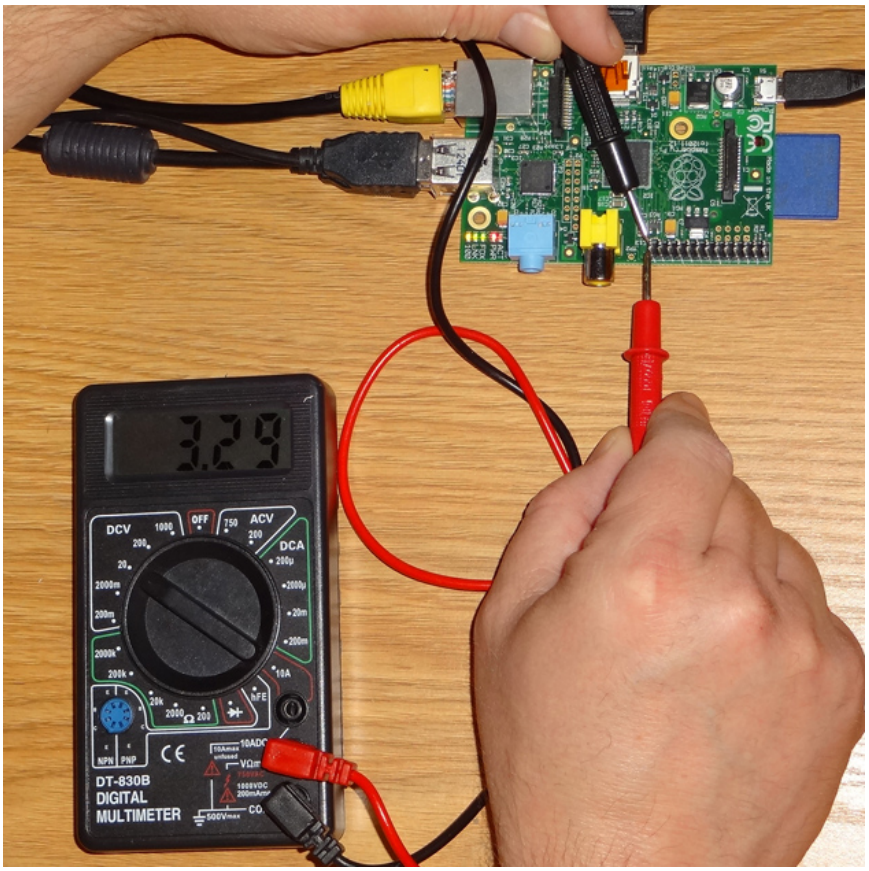


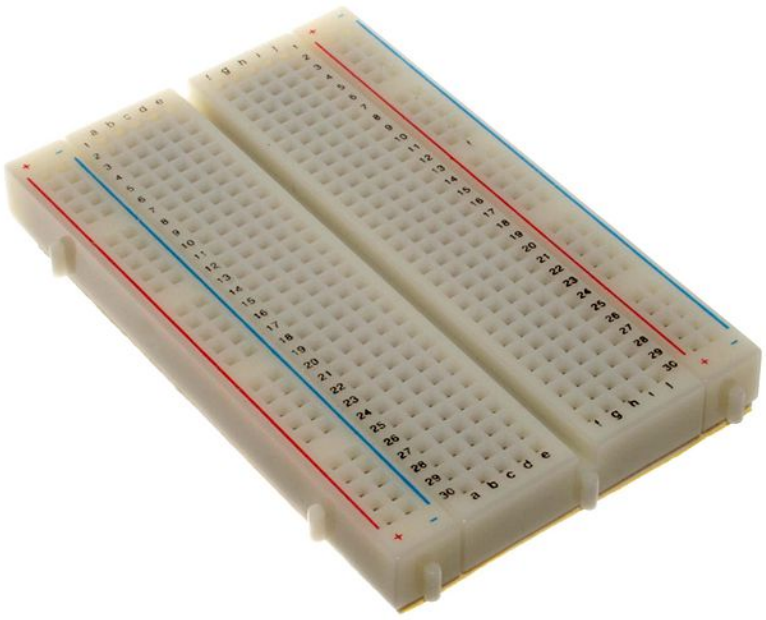
Figure 5–3 Raspberry Pi Pin 7 with High (3.3V) Voltage



Working with a Breadboard

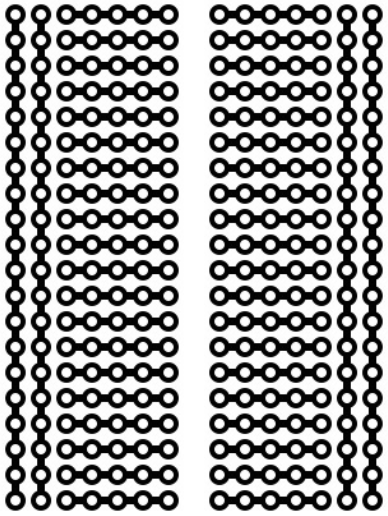
When prototyping circuits, it is often helpful to have a way of connecting wires without having to perform soldering. In some cases, if there are only a few connections, you can use jumper wires. However, when layout out more complex circuits, it's helpful to use a breadboard. A typical breadboard is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

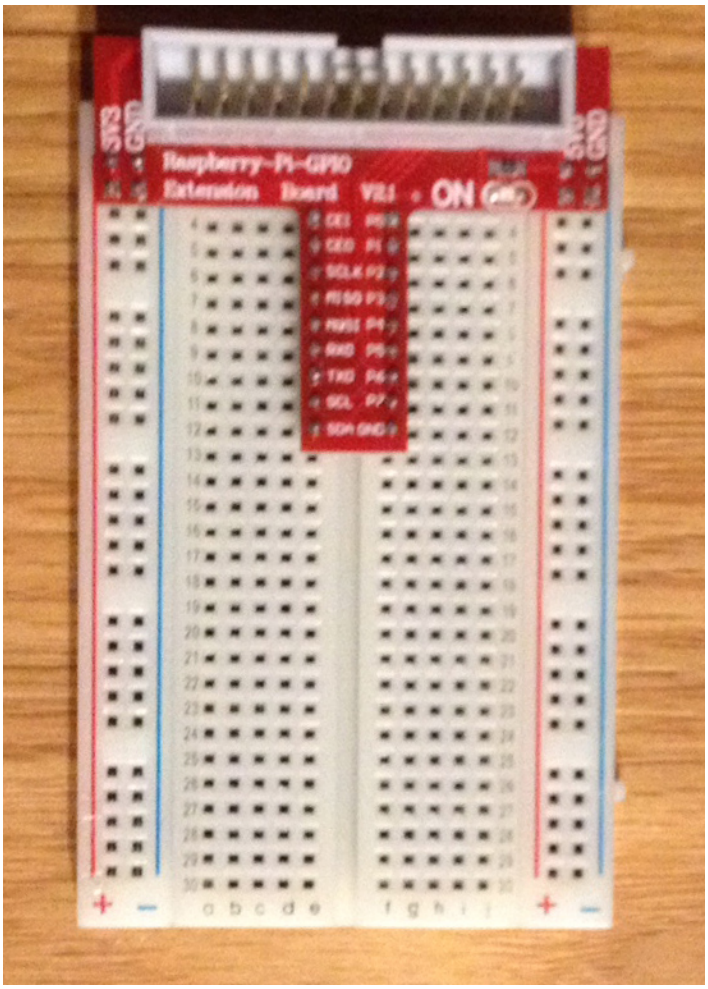
Figure 5–5 Wiring Pattern for a Typical Breadboard



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 Table 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

Table 5–3 (Cont.) Broadcom GPIO to T-Cobbler Conversion

GPIO (Pi Pin Number)	Alternate Name
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

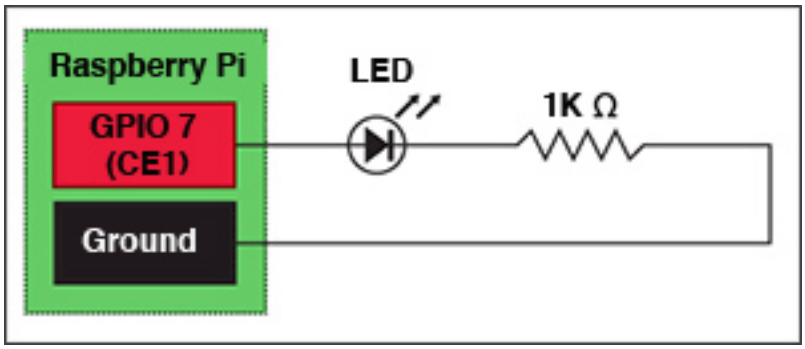
We can use the code in Example 5–1 to create a small circuit on the breadboard that turns on an off a light-emitting diode (LED). For this example, you will need the following equipment.

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.

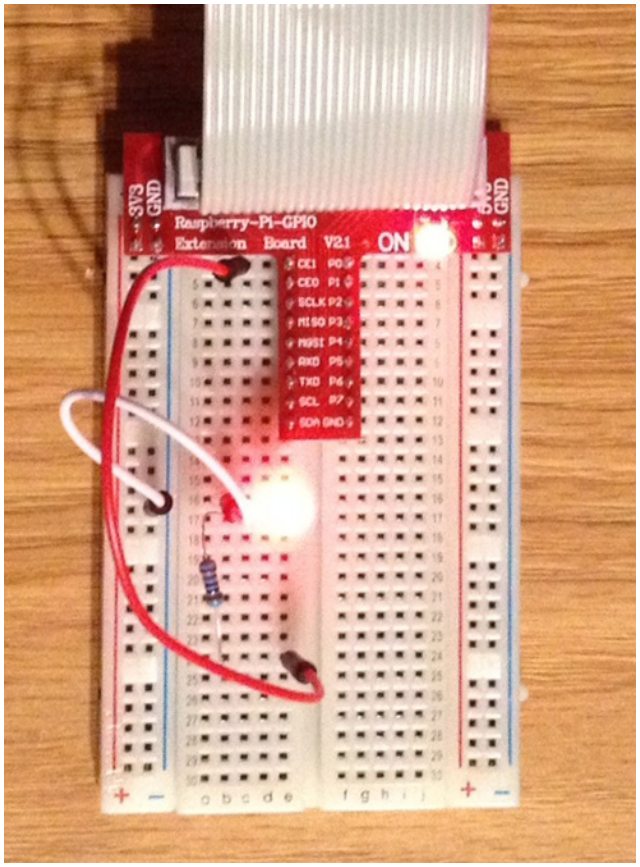
Figure 5–7 Schematic for Wiring an LED to GPIO 7



When completed, you should have a prototype that looks like Figure 5–8. Run Example 5–1 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 Example 5–2, 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).

Example 5–2 Creating a GPIO Pin Listener

```
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

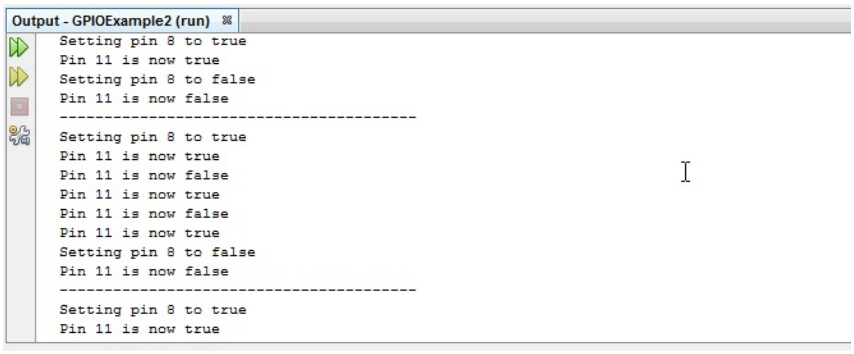
Permission	Device	Operation
`jdk.dio.DeviceMgmtPermission`	`*:*`	`open`

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	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
Ground (Pin 6)	GND

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

Pins on Raspberry Pi	HT16K33 Board
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. Example 6–1 shows the source code for the 7-segment I2C display driver.

Example 6–1 HT16K33 I2C Driver for 7-Segment Display

```
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.

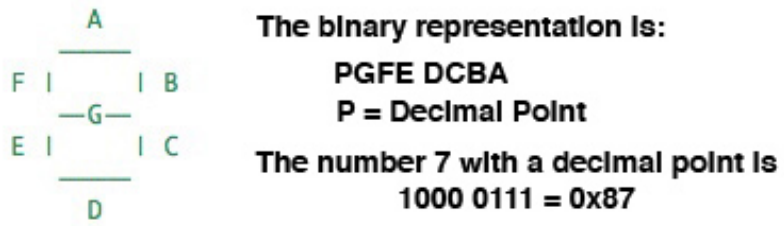
Each of the entries in the array will map to an address on the HT16K33 "LED backpack" that can be written to using the I2C bus. The purpose of each of the addresses is shown in Table 6–3. Note that since the HT16K33 can drive different types of LED displays, several of the addresses are ignored when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



Example 6–2 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).

Example 6–2 IMlet to Write to 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) {
    }
}
```

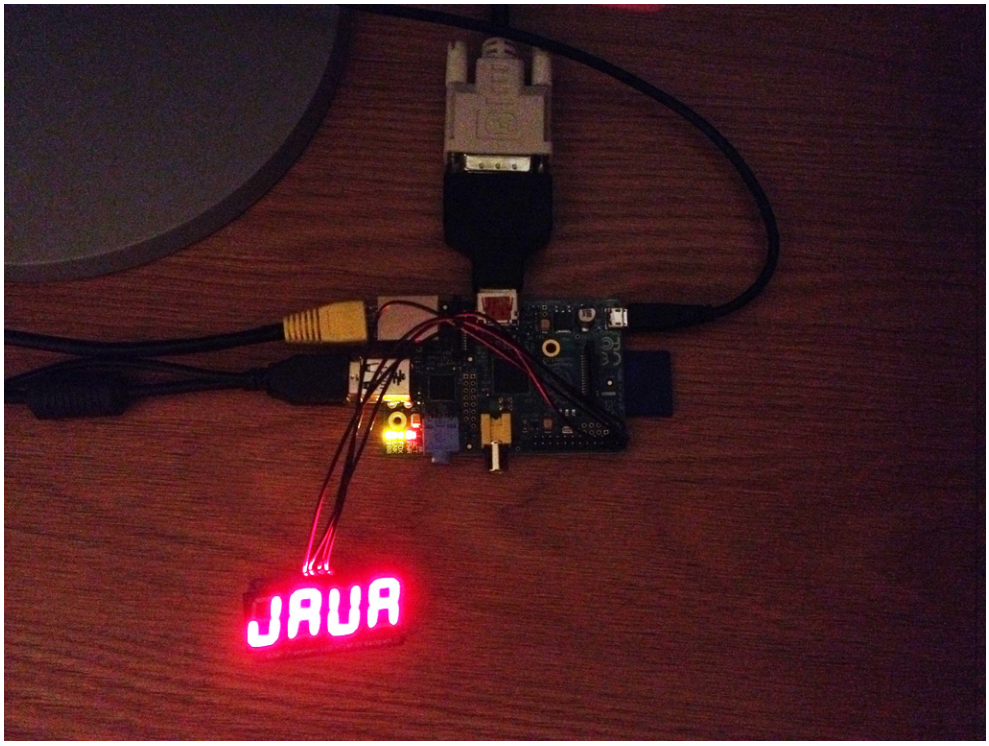
The following permissions must be added to the Application Descriptor of the project so that it will execute without any security exceptions from the Oracle Java ME Embedded runtime.

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:

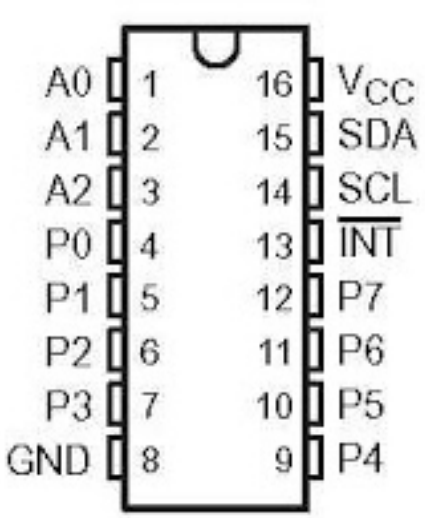
Table 6–5 Hardware for Example 2-2

Hardware	Where to Obtain
Raspberry Pi 512 MB Rev B	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



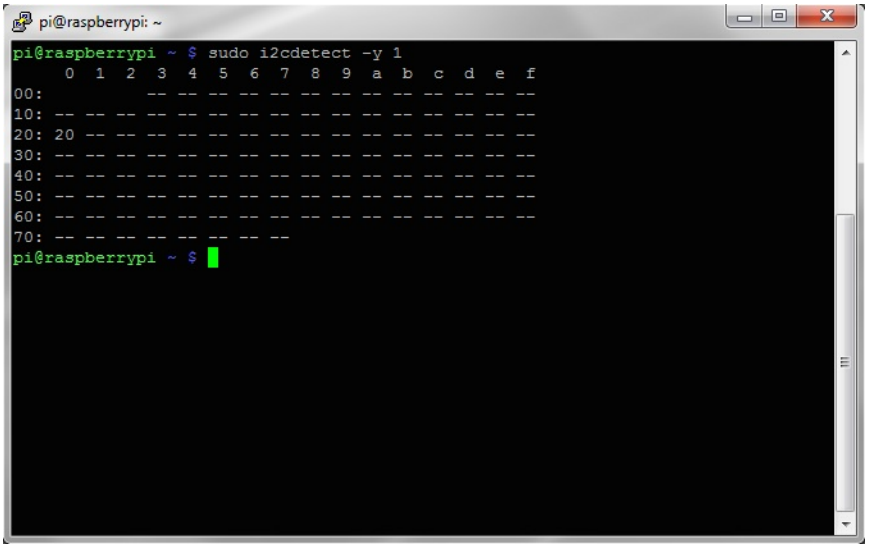
Once the chip is on the breadboard, there are several pins on the chip that must be connected to the T-Cobbler using jumper wires, as shown in Table 6–6.

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 from the PCF8574N chip and the HD44780 controller.

Table 6–7 Connections to PCF8574N and HD44780 Chip

Raspberry Pi (T-Cobbler)	PCF8574N	HD44780
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	
	P0	DB4
	P1	DB5
	P2	DB6
	P3	DB7
	P4	RS
	P5	R/W
	P6	E
	P7 (unused)	
	INT (unused)	
+5V (Pin 2)		VDD
0 to +5V		VO (variable resistor if desired for dimming backlit display)

Table 6–7 (Cont.) Connections to PCF8574N and HD44780 Chip

Raspberry Pi (T-Cobbler)	PCF8574N	HD44780
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 Example 6–3.

Example 6–3 Testing the PCF8574N I/O Expander Chip

```
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. Example 6–4 shows a sample driver class that will control the HD44780.

Example 6–4 LCD Driver Class to Control the HD44780 Chip

```
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 Example 6–5.

Example 6–5 IMlet to Write to the 16x2 LCD Display

```
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) {
    }
}
```

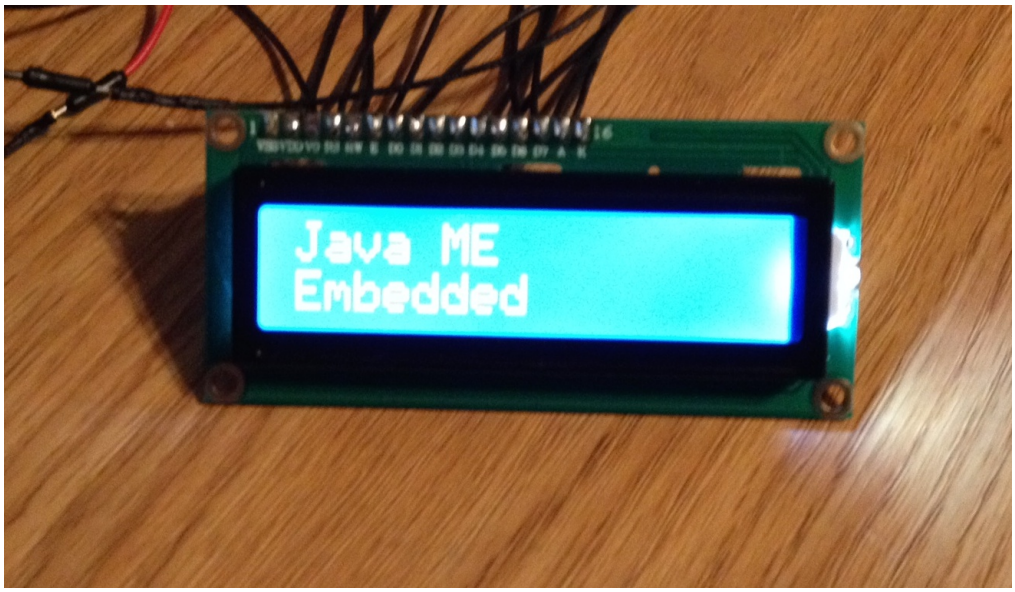
The following permissions must be added to the Application Descriptor of the project so that it will execute without any security exceptions from the Oracle Java ME Embedded runtime.

Table 6–8 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.

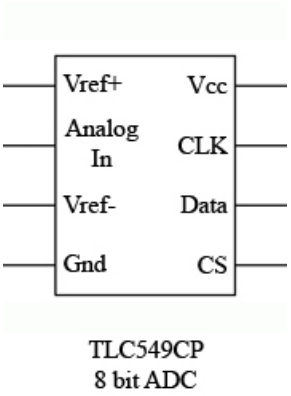
For this exercise, you will need the following hardware:

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.

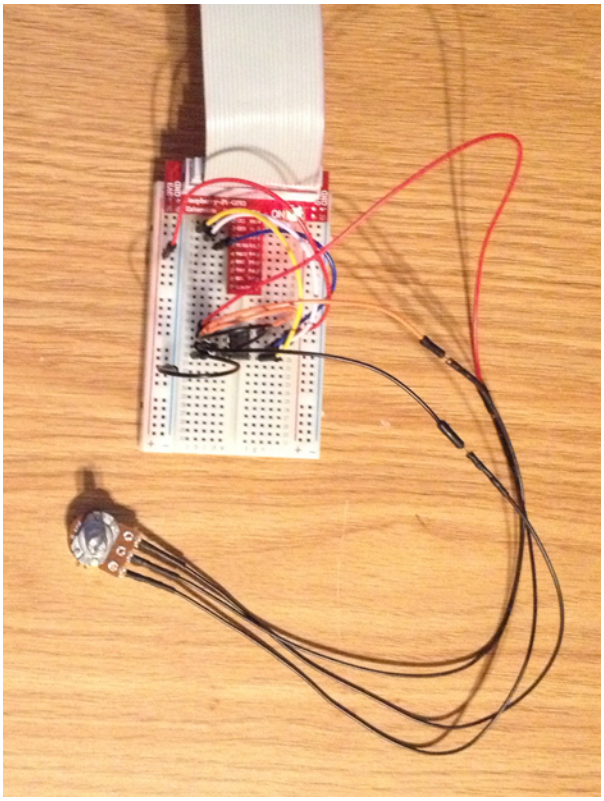
Table 7–3 shows how to connect the remaining pins on the TCL549CP chip.

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 Example 7–1 to test out the ADC chip.

Example 7–1 Testing Out the SPI Bus Connection

```
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
```

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.

Index