



Creating MIDlet Suites

MIDP Reference Implementation, Version 2.0 FCS

Java™ 2 Platform, Micro Edition

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, California 95054
U.S.A. 650-960-1300

November, 2002

Copyright © 2002 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents> and one or more additional patents or pending patent applications in the U.S. and in other countries.

This document and the product to which it pertains are distributed under licenses restricting their use, copying, distribution, and decompilation. No part of the product or of this document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Sun, Sun Microsystems, the Sun logo, Forte, Java, J2ME, Java Developer Connection, Java Powered logo, Javadoc, and J2SE are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.

The Adobe® logo is a registered trademark of Adobe Systems, Incorporated.

Federal Acquisitions: Commercial Software - Government Users Subject to Standard License Terms and Conditions.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright © 2002 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. a les droits de propriété intellectuels relatants à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et sans la limitation, ces droits de propriété intellectuels peuvent inclure un ou plus des brevets américains énumérés à <http://www.sun.com/patents> et un ou les brevets plus supplémentaires ou les applications de brevet en attente dans les Etats - Unis et dans les autres pays.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y ena.

Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Sun, Sun Microsystems, le logo Sun, Forte, Java, J2ME, Java Developer Connection, Java Powered logo, Javadoc, et J2SE sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.

Le logo Adobe® est une marque déposée de Adobe Systems, Incorporated.

LA DOCUMENTATION EST FOURNIE "EN L'ÉTAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFAÇON.



Contents

Preface ix

1. Introduction 1

Architecture 1

MIDlet Environment 2

MIDlet Life Cycle 5

Getting and Running MIDlets 6

Permissions 7

2. Creating a MIDlet 9

Getting Started 10

Constructing Objects 10

Handling User Input 12

Creating the Life Cycle Methods 13

APIs and Security 15

3. Compiling and Preverifying a MIDlet 17

Compiling MIDlet Source Files 18

Preverifying Compiled Files 19

Preverifying Class Files 19

Preverifying and Examining Class Files 20

4. Packaging a MIDlet	21
Creating a JAR File	22
Creating a JAD File	25
Signing a JAR File	26
5. Publishing a MIDlet	29
6. Debugging	31
Debugging a MIDlet	31
Reporting Problems	32
A. Code for the Hello MIDlet	35

Figures

FIGURE 1	J2ME Platform Architecture for MIDP and MIDP Applications	2
FIGURE 2	Screens of the PushPuzzle Game	3
FIGURE 3	Screen and Associated System Menu of Abstract Commands	4
FIGURE 4	Life Cycle of a MIDlet	5
FIGURE 5	Getting a MIDlet: Packaging to Installing	6
FIGURE 6	Hello World MIDlet	9
FIGURE 7	Building a MIDlet	17
FIGURE 8	Whitespace in an HTML File Shown on a Device	30

Code Samples

CODE EXAMPLE 1	Imports and Initial Class Definition for HelloMIDlet	10
CODE EXAMPLE 2	Constructor for HelloMIDlet	12
CODE EXAMPLE 3	Fields of HelloMIDlet	12
CODE EXAMPLE 4	The commandAction Method of HelloMIDlet	13
CODE EXAMPLE 5	Life Cycle Methods of HelloMIDlet	14
CODE EXAMPLE 6	Full Hello MIDlet Source Code	35

Preface

Creating MIDlet Suites describes how to create MIDlets and package them into MIDlet suites using the MIDP Reference Implementation. It is not a programming guide, so although it covers the required steps in MIDlet creation and discusses some interfaces, classes, and methods, it does not comprehensively cover the MIDP Reference Implementation APIs.

This guide assumes that you have already installed the product, as described in the *Installing MIDP*, and that you are familiar with both the Java™ programming language and the *MIDP 2.0 Specification*.

The Hello MIDlet is used as an example throughout this guide. The code for the Hello MIDlet is in the `midpInstallDir\src\example` directory, where `midpInstallDir` is the directory that holds your installation of the MIDP Reference Implementation. It is reproduced in [Appendix A, “Code for the Hello MIDlet.”](#)

How This Book Is Organized

This book has the following chapters:

[Chapter 1](#) introduces MIDlets and the environments in which they run.

[Chapter 2](#) describes the basics of creating a MIDlet.

[Chapter 3](#) provides the steps to compile and preverify your MIDlet.

[Chapter 4](#) shows you how to package your MIDlet.

[Chapter 5](#) describes how to publish your MIDlet.

[Chapter 6](#) describes how to run your MIDlet suite in a debugger and how to report problems.

[Appendix A](#) reproduces the example code for the Hello MIDlet.

Using Operating System Commands

This document may not contain information on basic UNIX[®] or Microsoft Windows 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.

Typographic Conventions

Typeface	Meaning	Examples
AaBbCc123	The names of commands, files, and directories; on-screen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. % You have mail.
AaBbCc123	What you type, when contrasted with on-screen computer output	% su Password:
<i>AaBbCc123</i>	Book titles, new words or terms, words to be emphasized	Read Chapter 6 in the <i>User's Guide</i> . These are called <i>class</i> options. You <i>must</i> be superuser to do this.
	Command-line variable; replace with a real name or value	To delete a file, type <code>rm filename</code> .

Shell Prompts

Shell	Prompt
C shell	%
Microsoft Windows	<i>directory></i>

Related Documentation

The following documentation is included with this release:

Application	Title
All	<i>Release Notes</i>
Installing	<i>Installing MIDP</i>
Running and managing security for emulator	<i>Using MIDP</i>
Porting the MIDP Reference Implementation	<i>Porting MIDP</i>
Creating and building MIDlets	<i>Creating MIDlet Suites</i>
Viewing reference documentation created by the Javadoc™ tool	<i>API Reference</i>
Looking at examples	<i>Example Overview</i>

Accessing Sun Documentation Online

The Java Developer Connectionsm web site enables you to access Java platform technical documentation on the Web:

<http://developer.java.sun.com/developer/infodocs/>

Sun Welcomes Your Comments

We are interested in improving our documentation and welcome your comments and suggestions. You can email your comments to us at:

docs@java.sun.com

Introduction

An application that runs in a MIDP environment is called a *MIDlet*. This chapter introduces MIDP Reference Implementation and MIDlets. It has the sections:

- [Architecture](#)
 - [MIDlet Environment](#)
 - [MIDlet Life Cycle](#)
 - [Getting and Running MIDlets](#)
 - [Permissions](#)
-

Architecture

MIDP (Mobile Information Device Profile) is part of the Java™ 2 Platform, Micro Edition (J2ME™). MIDP defines the Java application environment for mobile information devices (MIDs), such as mobile phones and personal digital assistants (PDAs). It is built on top of the Connected Limited Device Configuration (CLDC) and conforms to the specification from the Mobile Information Device Profile 2.0 [JSR-000118]. See <http://jcp.org/jsr/detail/118.jsp> for the *MIDP 2.0 Specification*.

A application that is written in the Java programming language and runs in the MIDP environment is a MIDlet. Users do not download and launch MIDlets though. They download and launch *MIDlet suites*, one or more MIDlets packaged together for distribution, then run a MIDlet from the suite.

MIDlet suites are typically made up of MIDlets that perform a similar function (such as a group of MIDlets in a game-pack) or that work together (such as a MIDlet that provides restaurant reviews and one that makes restaurant reservations). MIDlets in the same suite can share resources, such as graphics and data. If a MIDlet stores information on a device, other MIDlets in its suite can access the information, and other MIDlets can be given permission to see it.

The following figure shows the J2ME platform architecture from the MIDP perspective:

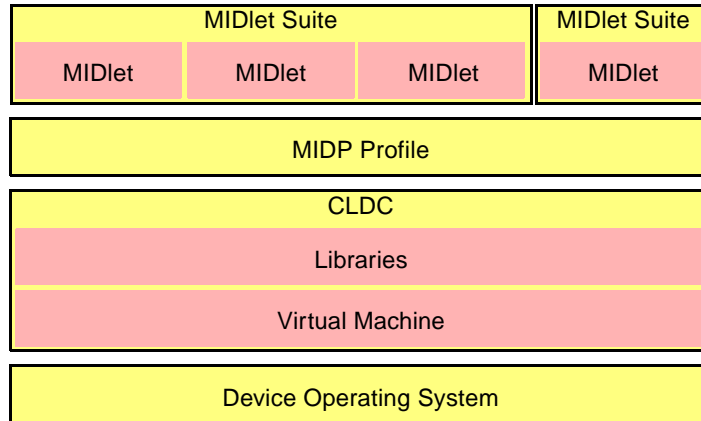


FIGURE 1 J2ME Platform Architecture for MIDP and MIDP Applications

MIDlet Environment

This section covers the MIDP environment from the perspective of the MIDlet developer. From this perspective, the MIDlet environment is screen based. That is, after determining the tasks that users will perform with a MIDlet, a developer organizes the tasks into screens. Users navigate through the screens when they run the MIDlet.

For example, a game such as PushPuzzle might enable a user to make a move in the game, undo last move, restart the level, restart the entire game, set the level of play, view high scores, and get information about the game. These tasks could be organized into:

- A screen for playing the game (making and undoing moves, restarting)
- A screen for setting the level of play
- A screen for seeing high scores
- A screen for seeing an “About box” for the application.

Here are the screens of the PushPuzzle application:

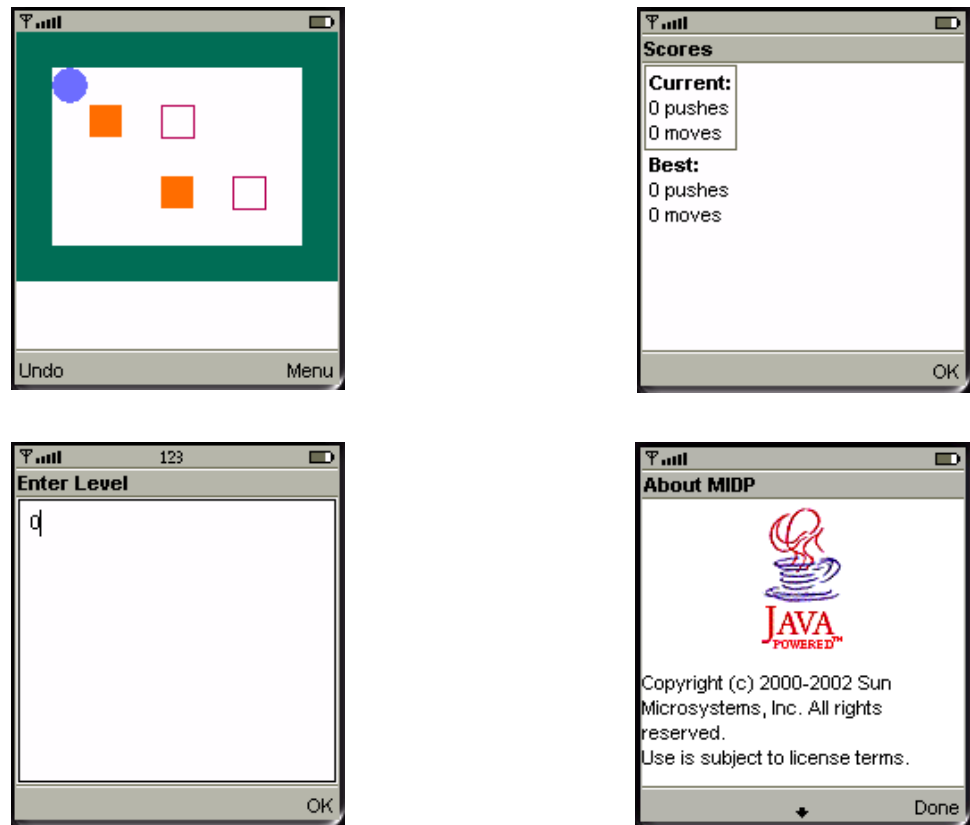


FIGURE 2 Screens of the PushPuzzle Game

MIDP has both *structured* and *unstructured* screens. Structured screens are more portable, but (with one exception, the `CustomItem` class) do not give the application access to low-level input mechanisms or control of the screen. Unstructured screens provides access to low-level I/O, but can be less portable. The screens in [FIGURE 2](#) show both types of screens: the game-playing screen is an unstructured screen; the rest are structured screens.

The unstructured screen is called a *canvas*. It gives you control of the screen, enabling you to draw images and simple graphics (such as rectangles and lines). It also gives you access to low-level input mechanisms, such as key presses or touch input if it is supported by the device. Again, a canvas is useful when you need control of the screen (such as for an action game) but is more difficult to make portable than a structured screen.

There are a few types of structured screens. Each type serves a particular purpose, such as gathering text input, alerting users to important events, or giving users lists of choices. When you use a structured screen you provide only the content, such as the elements in a list. The MIDP implementation handles the look of structured screens (such as layout, fonts, and colors), as well as their low-level interactions

with the user (such as scrolling). The MIDP implementation also notifies your application when the user takes an action, such as choosing OK on the screen in [FIGURE 2](#) titled Enter Level. Because MIDP implementations handle the user interfaces and IO, a MIDlet that uses structured screens can run on many devices and, without code updates, look and behave like a native application.

An action associated with a screen, such as OK on the screen titled Enter Level in [FIGURE 2](#), is called an *abstract command*. A screen can have any number of abstract commands; each screen should have at least one. Abstract commands enable you to define actions without specifying their user interface. MIDP implementations determine their presentation. This makes abstract commands portable, like structured screens. MIDlets that use them can look and behave appropriately on different devices without code changes.

When presenting abstract commands, MIDP implementations conform to the conventions of the device (within the constraints of the *MIDP 2.0 Specification*). Some implementations might use buttons, others menus, and so on. The MIDP Reference Implementation uses various buttons and, if there are more abstract commands after doing the standard mappings, creates a menu for them (a *system menu*). The following figure shows a screen that required a system menu, and the system menu that appears when the user chooses the Menu command.

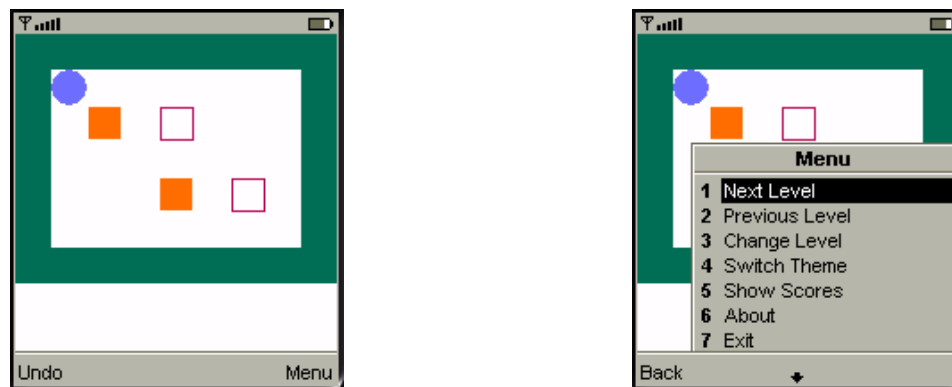


FIGURE 3 Screen and Associated System Menu of Abstract Commands

In summary, the MIDP environment is screen-based. When you design and implement a MIDlet, you need to organize its tasks into a set of screens. The screens can be either structured screens (which are more portable) or unstructured screens (if you need low-level I/O control). Each screen should have one or more associated actions, called abstract commands, that enable a user to carry out their tasks when they run the MIDlet.

MIDlet Life Cycle

The life cycle of a MIDlet consists of three states:

- Paused — Not in use, but available to become active when requested

A newly instantiated MIDlet is in the Paused state. A paused MIDlet probably holds fewer resources than an active MIDlet.

- Active — In use, interacting with the user through its screens
- Destroyed — Neither in use nor available

A destroyed MIDlet is eligible to be garbage collected.

A MIDlet can move between the paused and active states any number of times. It cannot move from the destroyed state.

The following figure shows the life-cycle of a MIDlet:

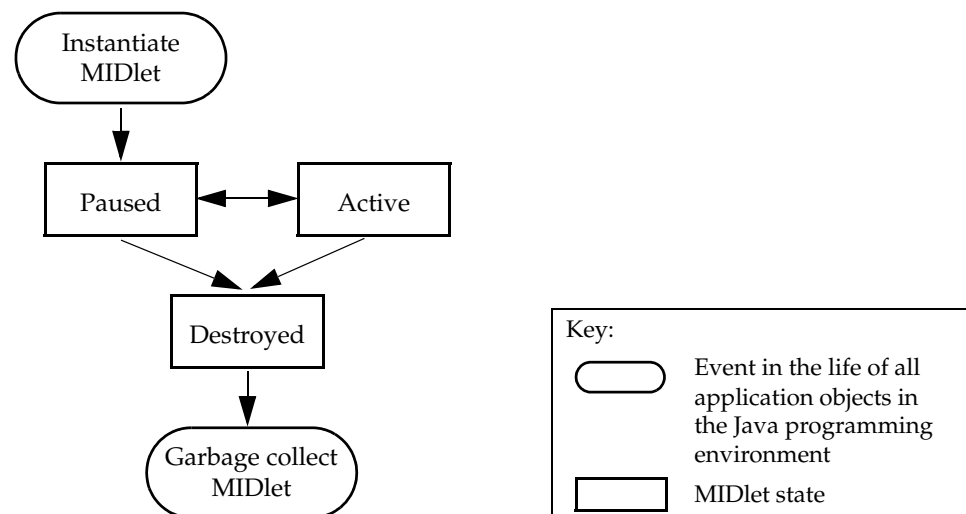


FIGURE 4 Life Cycle of a MIDlet

The next chapter, which tells you how to create a MIDlet, maps the MIDlet states onto the MIDlet's methods.

Getting and Running MIDlets

Before a MIDlet can be run, it must be put onto the device. From an end-user's perspective, that means downloading and installing a packaged MIDlet suite. (See *Using MIDP* for more information.) To enable users to get a MIDlet suite on the device, developers must package and publish the MIDlet. (These steps are covered in [Chapter 4, "Packaging a MIDlet,"](#) and [Chapter 5, "Publishing a MIDlet."](#))

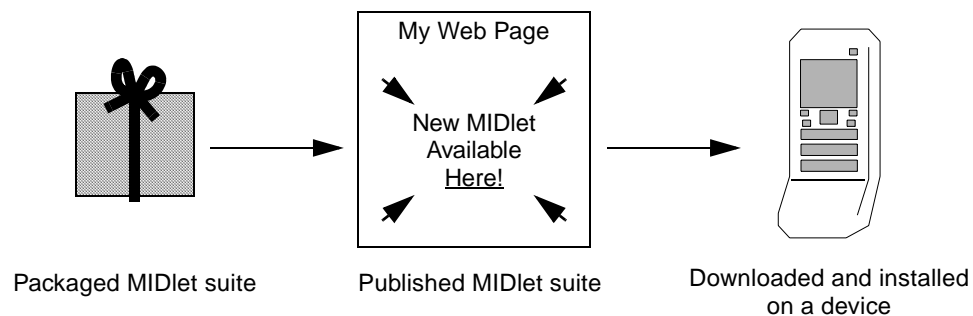


FIGURE 5 Getting a MIDlet: Packaging to Installing

Whether the device accepts the MIDlet that the user is trying to download and install depends on the *security policy* of the device. A security policy defines which security-sensitive actions MIDlet suites might be permitted to perform. If the MIDlet suite must perform an action that the device cannot allow, the device will not download and install the suite.

Security policies, and the ability to restrict access to security-sensitive APIs, is new to the *MIDP 2.0 Specification*. The specification also includes the concept of a *trusted* MIDlet suite, which is a MIDlet suite that originates from a trusted source and has not been tampered with. Typically a trusted MIDlet suite can perform more actions than an untrusted one.

The procedure for determining whether a MIDlet suite is trusted is device-dependent. The MIDP Reference Implementation checks the digital signature of the MIDlet suite. (See *Using MIDP* for more information.) Because checking a digital signature is one way of establishing trust, signing a suite when you package it will improve the chances that devices will trust your MIDlet suite. The security policies of individual devices, however, will determine whether particular users can download and effectively use your MIDlet suite.

Permissions

When a device defines its security policy, it uses *permissions*. A permission is a string that represents a security-sensitive class, package, or functionality. A security policy associates permissions with an interaction mode (such as Allowed). When you write a MIDlet, you need to be aware of which APIs are security sensitive; “APIs and Security” on page 15 lists them. When you package your MIDlet into a MIDlet suite, you need to list the permissions that your MIDlet would like to have and the permissions that it requires; “Creating a JAR File” on page 22 tells you how. This section explains how permissions are named.

The name of a permission indicates what it is protecting. If a permission is protecting a package or class, it has the package or class name. For example, a permission that protects the entire input/output package would be named `javax.microedition.io`, while a permission that protects the `PushRegistry` class would be called `javax.microedition.io.PushRegistry`. If a permission is protecting some functionality that a package or class provides, it has the package or class name as a prefix, followed by a name for the functionality. For example, a permission that protects the HTTP functionality made available through the `Connector` class would be called `javax.microedition.io.Connector.http`.

The *MIDP 2.0 Specification* defines the following permission names:

TABLE 1 Permission Names Defined in the *MIDP 2.0 Specification*

Functionality	Permission Name
HTTP protocol	<code>javax.microedition.io.Connector.http</code>
HTTPS protocol	<code>javax.microedition.io.Connector.https</code>
Datagram protocol	<code>javax.microedition.io.Connector.datagram</code>
Datagram server protocol (without host)	<code>javax.microedition.io.Connector.datagramreceiver</code>
Socket protocol	<code>javax.microedition.io.Connector.socket</code>
Server socket protocol (without host)	<code>javax.microedition.io.Connector.serversocket</code>
SSL protocol	<code>javax.microedition.io.Connector.ssl</code>
Comm protocol	<code>javax.microedition.io.Connector.comm</code>
Use the registry that holds information on MIDlets that use <i>push</i> functionality (that is, MIDlets that will be launched to handle incoming messages)	<code>javax.microedition.io.PushRegistry</code>

Creating a MIDlet

This chapter covers the basics of writing a MIDlet. It contains the sections:

- [Getting Started](#)
- [Constructing Objects](#)
- [Handling User Input](#)
- [Creating the Life Cycle Methods](#)
- [APIs and Security](#)

The Hello MIDlet example is shown in the following figure. The MIDlet has a single screen that has Hello MIDlet as its title. The screen is a text box that accepts edits and initially displays the string “Test string.” It has a single abstract command with the label Exit.



FIGURE 6 Hello World MIDlet

The code for the Hello MIDlet is in the *midpInstallDir*\src\example directory, where *midpInstallDir* is the directory that holds your installation of the MIDP Reference Implementation. The code is reproduced in [Appendix A, “Code for the Hello MIDlet.”](#)

Getting Started

MIDlets must extend the `MIDlet` class. They often also implement the `CommandListener` interface as a matter of convenience. The `CommandListener` interface contains the method that the MIDP implementation calls to notify your application that the user has selected an abstract command.

The `MIDlet` class is in the package `javax.microedition.midlet`. The `CommandListener` interface is in the `javax.microedition.lcdui` package, as are the structured screens, unstructured screens, and abstract commands.

Initially the file for the Hello World MIDlet could look like this:

CODE EXAMPLE 1 Imports and Initial Class Definition for HelloMIDlet

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

/**
 * An example MIDlet with simple "Hello" text and an Exit command.
 */
public class HelloMIDlet extends MIDlet implements CommandListener {
    // MIDlet will be defined here.
}
```

Constructing Objects

A MIDlet constructor should perform any setup that should be done only once in the lifetime of the MIDlet. Typically the constructor should:

- Instantiate the MIDlet's screens and abstract commands. Screens that may not appear, such as alerts, are not typically created at MIDlet instantiation. Instead, they are created only when they are needed.
- Associate screens with their abstract commands. All screens should have at least one abstract command. A single abstract command can be associated with multiple screens.
- Assign a listener to each screen. The listener is the class that implements the `CommandListener` interface; for the example it's the `HelloMIDlet` class.
- Set static or instance fields. A MIDlet class typically has fields that hold its screens and abstract commands.

The constructor can also set the MIDlet's display to the initial screen of the MIDlet. This task could also be done in the `startApp` method; see [“Creating the Life Cycle Methods” on page 13](#) for more information.

The Hello MIDlet uses a structured screen called a text box, which the MIDlet stores in an instance variable named `textBox`. Because the text box does not need to store any additional data or implement any additional functionality, it can be an instance of the `TextBox` class. (If it did require additional functionality, such as the ability to query a database to get its initial string, you would create a subclass of `TextBox` to instantiate.)

You can create text boxes with a title, initial text, the maximum number of characters the text box can hold, and a value that represents any *constraints* and *modifiers*. A constraint is a request that the MIDP implementation uses to limit user input. A modifier is a request that the MIDP implementation uses to change the behavior of text entry and display.

The Hello MIDlet's text box has the title "Hello MIDlet" and the initial text "Test string." It can hold up to 256 characters. It has the constraint `TextField.ANY`, which means the text box may contain any text; for example, it is not limited to holding only numbers. It also has the modifier `TextField.UNEDITABLE`, which means that the user is not permitted to change what is in the text box.

The Hello MIDlet also has a single abstract command that enables users to quit the application. When you create an abstract command you need three pieces of information:

- **Label** — A string that MIDP can show to the user
- **Type** — A constant that tells MIDP the category of action that the user will take with the command, so that MIDP can provide the correct user interface

For example, some devices have a key typically used to return to a previous state. The MIDP implementation might try to always assign a command of type `Command.BACK` to this key.

- **Priority** — A positive integer that identifies how important the command is compared to other commands of the same type on that screen; lower integers are more important

The Hello MIDlet's command is associated with the text box. The command in the Hello MIDlet has the label `Quit`, type `Command.EXIT`, and priority one.

[CODE EXAMPLE 2](#) shows the constructor for the Hello MIDlet.

CODE EXAMPLE 2 Constructor for HelloMIDlet

```
/**
 * Constructs the HelloMIDlet, its screen, and
 * its command. It associates the screen and command,
 * and caches the MIDlet's display.
 */
public HelloMIDlet() {
    // Create the abstract command
    exitCommand = new Command("Exit", Command.EXIT, 1);

    // Create the screen and give it a command and a listener
    textBox = new TextBox("Hello MIDlet", "Test string",
                          256,
                          TextField.ANY | TextField.UNEDITABLE);
    textBox.addCommand(exitCommand);
    textBox.setCommandListener(this);

    // Set the MIDlet's display to its initial screen
    display = Display.getDisplay(this);
    display.setCurrent(textBox);
}
```

The variables that it sets, `textBox`, `exitCommand`, and `display`, are declared prior to defining the class. The following code fragment shows the fields in the Hello MIDlet:

CODE EXAMPLE 3 Fields of HelloMIDlet

```
// Fields for the screens, commands, and display of this MIDlet
private TextBox textBox;
private Command exitCommand;
private Display display;
```

Handling User Input

The MIDP implementation notifies your application every time the user carries out an action (an abstract command) associated with a screen. It does this by calling the `commandAction` method of the `CommandListener` interface on the screen's listener.

The `commandAction` method takes as arguments the action that the user has chosen and the screen that was being displayed. (You need to know the screen because you can associate the same abstract command with multiple screens.)

In MIDlets set up like the example, with one class listening for commands, the `commandAction` method typically contains an `if` statement. For example, in natural language, "If the user chooses Quit, exit the MIDlet. If the user chooses Go

Back, then if the screen is ScreenB, set the display to ScreenA, if the screen is ScreenC, set the display to ScreenB, and so on.” The following code example shows the `commandAction` method for the Hello MIDlet:

CODE EXAMPLE 4 The `commandAction` Method of HelloMIDlet

```
/*
 * Responds to the user's selection of abstract commands.
 * This MIDlet has only an exit command, this method responds
 * by cleaning up and notifying the system that the MIDlet
 * has been destroyed.
 *
 * @param command the command the user has chosen
 * @param screen the displayable shown when the user chose the
 * command
 */
public void commandAction(Command command, Displayable screen) {
    if (command == exitCommand) {
        // Always make these two calls when exiting a MIDlet.
        destroyApp(false);
        notifyDestroyed();
    }
}
```

Creating the Life Cycle Methods

There are three life cycle methods in the `MIDlet` class. The methods correspond to the states in the MIDlet life cycle discussed in [Section “MIDlet Life Cycle” on page 5](#). The methods are:

- `startApp` — Notifies the MIDlet that it has just been moved from the Paused state to the Active state. If your MIDlet requires any shared resources when it runs, have this method acquire them. It can also set the initial screen if the MIDlet’s constructor hasn’t done that. It should not set subsequent screens.
- `pauseApp` — Notifies the MIDlet that it has just been moved from the Active state to the Paused state. If the MIDlet is holding shared resources, this method should release them. (The `startApp` method should reacquire them when the MIDlet becomes active again.) If the MIDlet should restart with a particular screen, this method should set it by calling the `setCurrent` method.
- `destroyApp` — Notifies the MIDlet that it should prepare to enter the Destroyed state by releasing its resources and saving any persistent state. Unlike the other life cycle methods, this method takes an argument indicating whether the MIDlet must be destroyed. If the argument is `false` (destruction is requested but not required), you can have the MIDlet request to be destroyed at a later time by having the method throw a `MIDletStateChangeException`. The MIDP system could, in response, not change the state of the MIDlet and call the `destroy` method again later.

The life cycle methods are declared abstract; you must implement them in your MIDlet subclass in order to instantiate your MIDlet. The following code sample shows the life cycle methods of the Hello MIDlet. Because the MIDlet does not use shared resources, the methods are empty.

CODE EXAMPLE 5 Life Cycle Methods of HelloMIDlet

```
/**
 * Starts the MIDlet; this method does nothing because
 * the MIDlet does not require any shared resources.
 */
public void startApp() {
}

/**
 * Pauses the MIDlet; this method does nothing because
 * there are no background activities or shared resources
 * to close.
 */
public void pauseApp() {
}

/**
 * Destroys the MIDlet; this method does nothing because
 * there is nothing to cleanup that is not handled by the
 * garbage collector.
 *
 * @param unconditional Whether the MIDlet must be destroyed.
 * If true, the MIDlet must cleanup and release all resources.
 * If false the MIDlet may throw a MIDletStateChangeException to
 * indicate it does not want to be destroyed at this time.
 */
public void destroyApp(boolean unconditional) {
}
```

APIs and Security

The MIDP 2.0 security model protects security-sensitive APIs. The APIs in the following table are security sensitive. If you use them in your MIDlet, you must make that known when you package it into a MIDlet suite. (See [“Creating a JAR File” on page 22](#) for instructions.)

TABLE 2 APIs That Require Permission

API	Description
<code>javax.microedition.io.HttpConnection</code>	API for making HTTP requests
<code>javax.microedition.io.HttpsConnection</code>	API for making secure HTTP requests
<code>javax.microedition.io.Connector</code>	API for creating connections for the protocols: <ul style="list-style-type: none">• HTTP• HTTPS• Datagram• Datagramreceiver• Socket• Serversocket• SSL• Comm
<code>javax.microedition.io.PushRegistry</code>	API for setting up the MIDlet for using <i>push</i> functionality (that is, setting up the MIDlet so that the device can launch it to handle incoming messages)

Not all APIs are security sensitive. The following table shows the APIs that all MIDlets can freely use:

TABLE 3 APIs That Do Not Require Permission

API	Description
<code>javax.microedition.rms</code>	API for using the persistent storage on the device
<code>javax.microedition.midlet</code>	The MIDlet class and other MIDlet life cycle APIs
<code>javax.microedition.lcdui</code>	User interface API
<code>javax.microedition.lcdui.game</code>	API for MIDlets that are games
<code>javax.microedition.media</code> and <code>javax.microedition.media.controls</code>	API for playing sound on the device

Compiling and Preverifying a MIDlet

Like all applications written in the Java™ programming language, you must compile a MIDlet before running it. In addition to compiling, you must preverify the MIDlet. The preverification step adds additional information to your `.class` files so that they can be run in the MIDP environment. (Preverified class files can still be run by the Java 2 Platform, Standard Edition.)

This chapter covers both steps. It contains the sections:

- [Compiling MIDlet Source Files](#)
- [Preverifying Compiled Files](#)

You will use the `.class` files from the preverifier when you run your MIDlet. Therefore, when you compile your MIDlet, put the compiled files into a temporary directory for the preverifier. Put the preverifier's output into a more permanent location, from which you can package or run the MIDlet. The following figure illustrates this sequence:

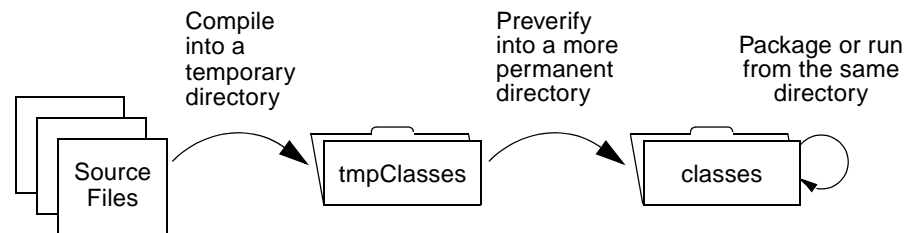


FIGURE 7 Building a MIDlet

Compiling MIDlet Source Files

To compile a MIDlet, use the usual compiler, `javac`. (See the *Release Notes* for version requirements.) Use the `-d` option to `javac` so that all the `.class` files are put into a single, temporary directory. This makes the rest of the build process easier.

For example, to create an output directory, `c:\midp2.0fcs\tmpclasses`, and compile the MIDlet into that directory, you would enter the following commands:

```
c:> cd midp2.0fcs
c:\midp2.0fcs> mkdir tmpclasses
c:\midp2.0fcs> javac -classpath classes -d tmpclasses
src/example/HelloMIDlet.java
```

There is an option for `javac` that you must use if your MIDlet is using the *push* functionality defined in the *MIDP 2.0 Specification*; the option is `-bootclasspath`. (Push functionality enables MIDP to launch a MIDlet to handle a message being sent to it. For example, if you are writing a MIDlet from which a user can get traffic reports, you might have the MIDlet use push to receive urgent warnings about the user's commute route.)

The `-bootclasspath` option must specify the location of the MIDP class files (*midpInstallDir/classes*) because of differences in the class hierarchies of the J2ME and J2SE platforms. For example, the J2SE platform's `FilterInputStream` class is not part of the J2ME platform. If the MIDlet were compiled with J2SE classes, it would fail preverification (the next step in the build process). With the `-bootclasspath` option, `javac` uses the J2ME platform class files.

The following example shows the `javac` command and the `-bootclasspath` option being used to compile a MIDlet called `PushExample`:

```
c:\myMIDlet> javac -bootclasspath c:/midp2.0fcs/classes
-classpath c:/midp2.0fcs/classes PushExample.java
```

Note that the command uses both the `-classpath` option and the `-bootclasspath` option. The `-classpath` option gives the `javac` command places to search for classes that are not in the J2SE platform, such as classes in the `javax.microedition` package. It does not effect which platform (J2SE or J2ME) is used. The `-bootclasspath` option specifies the location of the Java platform files.

Preverifying Compiled Files

The preverifier creates new `.class` files that contain additional information that the MIDP Reference Implementation implementation requires. If a class file has not been preverified, MIDP will not be able to use it. (Preverified class files can still be used with a Java 2 Runtime Environment (JRE).) The preverifier can also check the class files to make sure that they do not use any Java programming language features that are prohibited by the CLDC specification. (See <http://jcp.org/jsr/detail/30.jsp> for the CLDC specification.)

The preverifier is in the `midpInstallDir\bin` directory; the executable name is `preverify`. Using the command with no options provides help. For example, you could get help with the command:

```
c:\midp2.0fcs\bin> preverify
```

For a summary of the command in a man page format, see *Using MIDP*.

Preverifying Class Files

The simplest way to run the preverifier is to have it create new `.class` files without checking for the use of prohibited features. You might want to use the preverifier this way if you recompile after making a simple bug fix.

To run the preverifier this way, provide the classes and input files to be preverified. The input files can be directories that contain `.class` files, or JAR or ZIP files that contain `.class` files. If you provide a directory, the preverifier also checks its subdirectories. The following example shows the `preverify` command being used to check all the `.class` files in the `tmpclasses` directory and its subdirectories:

```
c:\midp2.0fcs> bin\preverify -classpath classes;tmpclasses
tmpclasses
```

By default, the preverifier writes its new class files to the `./output` directory. You can change this directory, as shown in the following example, which preverifies a class and puts the preverified `.class` file into the `classes` directory

```
c:\midp2.0fcs> bin\preverify -classpath classes;tmpclasses
-d classes HelloMIDlet
```

After you run the preverifier, the contents of the output directory correspond to the types of *inputFiles* you provide. The directory will contain a `.class` file that corresponds to each input class, a directory tree that corresponds to each input directory, and a JAR file that corresponds to each input JAR file.

Preverifying and Examining Class Files

To both check for the use of Java programming language features that are not part of the CLDC specification and preverify your class files, you need to use one or more additional arguments to the `preverify` command. The additional arguments can check for the presence of floating point operations, the use of finalizers, and calls to native methods from application classes. The following example checks for these problems:

```
c:\midp2.0fcs> bin\preverify -nofp -nofinalize -nonative  
-classpath classes;tmpclasses -d classes HelloMIDlet
```

The following example is a simpler version of the previous one. It uses a single switch that is equivalent to using the `-nofp`, `-nofinalize`, and `-nonative` switches together:

```
c:\midp2.0fcs> bin\preverify -cldc  
-classpath classes;tmpclasses -d classes HelloMIDlet
```


Packaging a MIDlet

Packaging a MIDlet into a MIDlet suite prepares it to be downloaded and run. A MIDlet suite consists of two files: a JAR file, which holds the MIDlet classes and other resources (such as graphics), and a Java™ application description (JAD) file, which holds attributes and values that describe the MIDlet suite and its MIDlets.

In addition to creating the JAR and JAD file, you can optionally sign the JAR file. This gives the MIDlet a better chance of being trusted. (See [Chapter 1](#), “[Introduction](#)” for more information on trusted MIDlets.)

This chapter shows you how to package one or more MIDlets into a MIDlet suite. It has the sections:

- [Creating a JAR File](#)
- [Creating a JAD File](#)
- [Signing a JAR File](#)

Creating a JAR File

A JAR file is a bundle of files that includes the MIDlet's preverified files and other supporting files, such as graphics. You create the JAR file with the `jar` command. (The `jar` command is part of the Java 2 Platform, Standard Edition (J2SE™ platform) distribution. See the *Release Notes* for J2SE platform version requirements, and your Java platform documentation for more information on the `jar` command and JAR files.)

A JAR file for distributing a MIDlet suite must contain:

- A manifest file
- Class files for the MIDlets in the suite
- Resource files (such as graphics files) for the MIDlets in the suite

To create a JAR file for a MIDlet suite:

- 1. Ensure that all the preverified class files for the MIDlet suite are in their expected directory.**

For example:

```
c:\midp2.0fcs> dir classes
Volume in drive C has no label.
...
10/21/2002  03:32p                1,555 HelloMIDlet.class
...
```

- 2. Copy any resource files for the MIDlets into the directory that contains the preverified class files.**

The example MIDlet has no resource files. If, for example, it required a graphics file, `mySplashImage.png`, a command like the following one would put it in the same directory as the preverified files:

```
c:\midp2.0fcs> cp src/example/mySplashImage.png classes/
```

- 3. Decide what you will call the JAR file.**

The rest of this chapter will refer to it as *jarFileName*.

For example, the *jarFileName* for the MIDlet example will be `HelloMIDlet`.

- 4. Change directories to the location of the preverified class files and resource files.**

For example,

```
c:\midp2.0fcs> cd classes
```

5. Create a manifest file.

The manifest file, *filename.mf*, is a text file that you can create with any text editor. It must contain at least the following attributes:

- MIDlet-Name — Name that will be presented to the user
- MIDlet-Version — Version number of the MIDlet in the major.minor.micro format described in the Java Product Versioning Specification at <http://java.sun.com/products/jdk/1.2/docs/guide/versioning/spec/VersioningSpecification.html>
- MIDlet-Vendor — Creator of the MIDlet.

If you have used any security-sensitive APIs, you can also declare them in the manifest file. (If they are not in this file, they must be in the JAD file.) To declare them, use the following attributes:

- MIDlet-Permissions – Permissions that are critical to the correct functioning of the MIDlet suite
- MIDlet-Permissions-Opt – Permissions that the MIDlet suite would like, but that it can function correctly without (For example, use this if the MIDlet suite could run with reduced functionality if the permission were not available.)

The values for MIDlet-Permissions and MIDlet-Permissions-Opt are comma-separated lists. Leading and trailing white space (Unicode U+0020) and tabs (Unicode U+0009) are ignored. The following table shows the correspondence of MIDP 2.0 functionality and permissions:

TABLE 4 APIs and Their Corresponding Permission

API	Corresponding Permission Name
javax.microedition.io. HttpConnection	javax.microedition.io.Connector.http
javax.microedition.io. HttpsConnection	javax.microedition.io.Connector.https
javax.microedition.io. Connector	Use the permission corresponding to the protocol you used when you created a connection: <ul style="list-style-type: none">• javax.microedition.io.Connector.http• javax.microedition.io.Connector.https• javax.microedition.io.Connector.datagram• javax.microedition.io. Connector.datagramreceiver• javax.microedition.io.Connector.socket• javax.microedition.io.Connector.serversocket• javax.microedition.io.Connector.ssl• javax.microedition.io.Connector.comm
javax.microedition.io. PushRegistry	javax.microedition.io.PushRegistry

The manifest file can also contain the following attributes. (If they are not in this file, they must be in the JAD file.)

- `MIDlet-n` — Comma-separated list of the *n*th MIDlet's name, icon, and class where *n* is the *n*th MIDlet in the JAR file. The lowest value of *n* is one, and *n* increments by one for each consecutive MIDlet. Supplying an icon is optional.
- `MicroEdition-Profile` — J2ME platform profiles that the MIDlet requires. Multiple profiles are separated with a blank (Unicode x20). For MIDP 2.0, the value should be MIDP-2.0.
- `MicroEdition-Configuration` — J2ME platform configuration that the MIDlet requires. Multiple configurations are separated with a blank (Unicode x20). For MIDP 2.0, the value should be CLDC-1.0.

The manifest file is also permitted to contain other attributes. For more information on the legal attributes in a MIDlet's a JAR file, see the specification for the Mobile Information Device Profile 2.0 [JSR-000118] at <http://jcp.org/jsr/detail/118.jsp>.

For example, the manifest file for the `HelloMIDlet.jar` file, called `HelloMIDlet.mf`, looks like this:

```
MIDlet-Name: HelloWorld
MIDlet-Version: 2.0
MIDlet-Vendor: Sun Microsystems, Inc.
```

(If you are following this example by creating the Hello MIDlet, you should know that the `HelloMIDlet.mf` file is shipped in the `midpInstallDir\src\example` directory. Instead of entering a new version of the file, you can copy it into the `midpInstallDir\classes` directory.)

6. Create the JAR file with the `jar` tool.

Put the manifest file, class files, and resource files into the JAR file. (See your Java platform documentation for more information on the `jar` command.) For example:

```
c:\midp2.0fcs\classes> jar -cmf HelloMIDlet.mf HelloMIDlet.jar
HelloMIDlet.class
```

Creating a JAD File

A JAD file is a text file that you create with any text editor. Its lines have the following syntax:

attribute: value

The JAD file must contain at least the following attributes:

- MIDlet-Name — Same as the attribute in the JAR file's manifest. See [Step 5](#) in the previous section.
- MIDlet-Version — Same as the attribute in the JAR file's manifest. See [Step 5](#) in the previous section.
- MIDlet-Vendor — Same as the attribute in the JAR file's manifest. See [Step 5](#) in the previous section.
- MIDlet-Jar-URL — Location of the JAR file.
- MIDlet-Jar-Size — Number of bytes in the JAR file.

The JAD file may also contain the other attributes, including those listed in [Step 5](#) in the previous section. For more information on the legal attributes in a MIDlet's a JAD file, see the specification for the Mobile Information Device Profile 2.0 [JSR-000118] at <http://jcp.org/jsr/detail/118.jsp>.

The JAD and JAR file must have the same values for the MIDlet-Name, MIDlet-Version, and MIDlet-Vendor attributes. If the values differ, MIDP will not be able to load the MIDlet.

If the JAD and JAR file have other attributes in common, the requirements depend on whether the MIDlet suite is trusted:

- Trusted – The values of duplicated attributes must be identical or the MIDP Reference Implementation will not be able to load the MIDlet.
- Not trusted – The values of duplicated attributes may differ. If they differ, MIDP uses the values in the JAD file.

For example, the JAD file for the Hello MIDlet, `HelloMIDlet.jad`, could look like this:

```
MIDlet-Name: HelloWorld
MIDlet-Version: 2.0
MIDlet-Vendor: Sun Microsystems, Inc.
MIDlet-Jar-URL: HelloMIDlet.jar
MIDlet-Jar-Size: 1212
MicroEdition-Profile: MIDP-2.0
MicroEdition-Configuration: CLDC-1.0
MIDlet-1: Hello World,, HelloMIDlet
```

Signing a JAR File

Establishing trust is important for MIDlet suites that use security-sensitive APIs because trusted MIDlets can typically access more protected functionality than untrusted ones. Checking the signature of a MIDlet suite is one way that a device can determine whether to trust a MIDlet suite. As a result, signing a MIDlet suite makes it more likely that your MIDlet suite will be trusted by a device. (See *Using MIDP* for more information on signatures and certificates.)

The HelloMIDlet does not access protected APIs, so it is not necessary to sign its MIDlet suite. This section signs the MIDlet suite just to show how it is done.

The example below signs the MIDlet suite the RSA key pair provided with this release. The key pair is in the file `midpInstallDir\bin\j2se_test_keystore.bin`, which is a keystore that can be managed with the J2SE platform's `keytool` utility. (See <http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html> for more information on the `keytool` utility.) The password for the file is `keystorepwd`. The alias of the key pair is `dummyca` and its private-key password is `keypwd`. The file is provided for testing purposes.

For MIDlets that you will publish, you will probably use an RSA key pair backed by a certificate or certificate chain from a certificate authority. The certificate authority vouches for your public/private key pair. You must have imported that certificate or certificate chain into a J2SE keystore with the J2SE platform's `keytool` utility.

1. Add the certificate for your public key pair to the JAD file using the `JadTool` utility.

The `JadTool` utility will add the certificate as the value of an attribute named `MIDlet-Certificate-m-n` where *m* is the number of the certificate chain (it defaults to one; you can provide a different number with the `-chainnum` switch), and *n* is an integer that, for new certificates, begins at one and increments by one each time you add a new certificate to the JAD file.

For example, the following command would add the certificate as the value of the attribute `MIDlet-Certificate-1-1` to the JAD file:

```
c:\midp2.0fcs\classes> java -jar ../bin/JadTool.jar -addcert
                        -keystore ../bin/j2se_test_keystore.bin -alias dummyca
                        -storepass keystorepwd -inputjad HelloMIDlet.jad
                        -outputjad HelloMIDlet.jad
```

2. Optionally, verify that the certificate was added to the JAD file by using the `JadTool` utility to list the certificate in the JAD file.

```
c:\midp2.0fcs\classes> java -jar ../bin/JadTool.jar -showcert
                        -certnum 1 -inputjad HelloMIDlet.jad
Subject: C=US, ST=CA, L=Santa Clara, O=dummy CA, OU=JCT, CN=thehost
Issuer : C=US, ST=CA, L=Santa Clara, O=dummy CA, OU=JCT, CN=thehost
Serial number: 3d3ece8a
Valid from Wed Jul 24 08:58:02 PDT 2002 to Sat Jul 21 08:58:02 PDT
2012
Certificate fingerprints:
    MD5: 87:7f:5e:64:c8:dd:b4:bf:35:39:76:87:99:9b:68:82
    SHA: 9d:c0:88:ce:08:83:cd:e6:fe:13:8b:26:f6:b4:df:e2:da:3c:25:98
```

3. If you have a key pair backed by a certificate chain, import the intermediate certificates.

Import the intermediate certificates using the `JadTool` utility with the `-addcert` switch shown in [Step 1](#), taking care to use the correct chain order. That is, if the Acme company provides a certificate that vouches for your key pair, and the WidgetCertificates company vouches for the Acme certificate, and, finally, Verisign vouches for the WidgetCertificates certificate, you would import the Acme certificate followed by the WidgetCertificate. The Acme certificate would be `MIDlet-Certificate-1-2` and the WidgetCertificate certificate would be `MIDlet-Certificate-1-3`.

Note that you do not import the certificate of the *root* CA (that is, the certificate at the end of the chain, which is the certificate from Verisign in this example). The CA's public key will be on the device.

4. Sign the JAR file using the `JadTool` utility.

The `JadTool` utility will sign the JAR file, base64 encode the signature, and store it as the value of the `MIDlet-Jar-RSA-SHA1` attribute of the output JAD file.

Note – Be sure that the key you use to sign the JAR file is from the same JCA keystore entry as key pair you specified in [Step 1](#). The `JadTool` utility does not check that you are signing the JAR file with a keystore entry that has a certificate in the JAD file.

For example:

```
c:\midp2.0fcs\classes> java -jar ../bin/JadTool.jar -addjarsig
                        -keystore ../bin/j2se_test_keystore.bin -alias dummyca
                        -storepass keystorepwd -keypass keypwd
                        -jarfile HelloMIDlet.jar -inputjad HelloMIDlet.jad
                        -outputjad HelloMIDlet.jad
```


Publishing a MIDlet

Publishing a MIDlet suite makes it available on a web server for users to download. To do this:

1. **Ensure that your web server is configured to deliver specific MIME types for the JAD and JAR files.**

The MIME type for a JAD file is:

`text/vnd.sun.j2me.app-descriptor`

The MIME type for a JAR file is:

`application/java-archive`

See the documentation for your web server for configuration instructions.

2. **Put the JAD and JAR files in a directory that your web server can access.**
3. **Create an HTML file that contains a link to your JAD file.**

The HTML file gives users a way to find and download your MIDlet. Finding and downloading a MIDlet is called *over the air provisioning (OTA)*.

The HTML file does not require any particular name or layout. When users run the `midp` executable to find and download MIDlets, they can enter any URL and MIDP will display only the links to JAD files at that URL. It ignores the rest of the HTML page.

For example, an HTML page that lists the Hello World MIDlet could be called `get-hello.html` and be as simple as this:

```
<html>
  <head><title>The Hello MIDlet</title></head>
  <body>
    <a href="http://localhost:8080/midlets/HelloMIDlet.jad"
      >Hello MIDlet<a>
  </body>
</html>
```

Note that, although no particular format for the HTML file is required, the whitespace in the file can affect the layout of the MIDlet suite name on the device. For example, if the HTML file with a link to the “Hello MIDlet” has a carriage return between the words Hello and MIDlet, the name looks like this on the device emulator:



FIGURE 8 Whitespace in an HTML File Shown on a Device

4. Put the HTML file into a web server visible directory.
5. Test the link from the HTML page to the JAD file, and from the JAD file to the JAR file.

One way to do this is to run the device emulator. See *Using MIDP* for instructions

Debugging

MIDP Reference Implementation provides Java™ platform source-level debugging through the CLDC debugging architecture. See your CLDC documentation for a thorough description of the debugging architecture.

This chapter briefly describes the steps that you would take to use the architecture and report any problems. It contains the sections:

- [Debugging a MIDlet](#)
- [Reporting Problems](#)

Debugging a MIDlet

Before you can debug a MIDlet, you must have versions of the MIDP executable and the MIDlet that have debugging symbols in their class files. To see whether you have an acceptable version of the `midp` executable, run the `midp` command with the `-help` option. If the executable has Java programming language debugging capabilities, you will see the `-debugger` option listed. For example:

```
C:\midp2.0fcs> bin\midp -help
Usage: midp [<options>]
        Run the Graphical MIDlet Suite Manager.
...
or midp [<options>] -debugger ...
```

If the version of the `midp` executable that you are using does not support Java programming language debugging, see *Porting MIDP* for instructions on building one.

To create a version of the MIDlet that contains debugging symbols, use the `-g` option to the `javac` command. See your Java platform documentation for more information.

To debug a MIDlet:

1. **Open a command prompt or terminal window.**

2. **Change your current directory to *midpInstallDir*.**

For example, if the MIDP Reference Implementation were installed in the directory `c:\midp2.0fcs` you could run the command:

```
c:\> cd midp2.0fcs
```

3. **Start the MIDP Reference Implementation executable in debug mode.**

Use the `midp` command with the switches `-debugger` and `-port`. The port number should be 2800, the port number on which the KVM debug proxy expects the debugger to be running. See the KVM documentation for more information on the debug proxy, and *Using MIDP* for more information on using the MIDP Reference Implementation executable.

For example:

```
c:\midp2.0fcs\> bin\midp -debugger -port 2800 -classpath classes
example.pushpuzzle.PushPuzzle
```

4. **Start the KVM debug proxy.**

See the KVM documentation for information on correct syntax, arguments, and options. For example, the following command has the KVM debug proxy connect to the `midp` executable that you started in the previous step and then listen at port 5000 for software compliant with the Java™ Platform Debugger Architecture:

```
c:\midp2.0fcs\> java -jar c:/kvm/bin/kdp.jar kdp.KVMDebugProxy -l
5000 -p -r localhost 2800 -cp pathsIncludingMIDletClassFiles
```

5. **Connect to the KVM debug proxy from any debugger compliant with the Java Platform Debugger Architecture.**

Compliant debuggers include `jdb`, Sun™ ONE Studio (formerly known as Forte™ for Java), JBuilder, Code Warrior, Visual Cafe, and others. See your debugger's documentation for information on attaching to a running VM on the localhost at port 5000.

Reporting Problems

Send any comments, questions or general feedback to `midp-comments@sun.com`. If you think you have found a bug, first check the bug parade to see whether the problem has already been reported. The bug parade is at this URL:

<http://developer.java.sun.com/developer/bugParade/index.jshtml>

The category for this product is Mobile Information Device Profile (MIDP) Reference Implementation.

If no one else has reported the bug, report it so that it can be reproduced, analyzed, and fixed. To do this, gather the following information:

- All platform and variant information, if not built on the standard platform
- Description of the bug
- Any error messages, exceptions, stack traces, or other data that you obtained
- If possible, a script or code fragment that demonstrates the problem

Enter the information in a bug report at this URL:

<http://java.sun.com/cgi-bin/bugreport.cgi/>

Code for the Hello MIDlet

The following is the source code for the Hello MIDlet:

CODE EXAMPLE 6 Full Hello MIDlet Source Code

```
/*
 * @(#)HelloMIDlet.java 1.12 02/09/25 @(#)
 *
 * Copyright (c) 2000-2002 Sun Microsystems, Inc. All rights reserved.
 * PROPRIETARY/CONFIDENTIAL
 * Use is subject to license terms.
 */

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

/**
 * An example MIDlet with simple "Hello" text and an exit command.
 */
public class HelloMIDlet extends MIDlet implements CommandListener {

    // Fields for the screens, commands, and display of this MIDlet
    private Command exitCommand;
    private TextBox textBox;
    private Display display;
```

```

/**
 * Constructs the HelloMIDlet, its screen, and its command. It
 * associates the screen and command, and caches MIDlets display.
 */
public HelloMIDlet() {
    // Create the abstract command
    exitCommand = new Command("Exit", Command.EXIT, 1);

    // Create the screen and give it a command and a listener
    textBox = new TextBox("Hello MIDlet", "Test string", 256,
        TextField.ANY | TextField.UNEDITABLE);
    textBox.addCommand(exitCommand);
    textBox.setCommandListener(this);

    // Set the MIDlet's display to its initial screen
    display = Display.getDisplay(this);
    display.setCurrent(textBox);
}

/**
 * Starts the MIDlet; this method does nothing because
 * the MIDlet does not require any shared resources.
 */
public void startApp() {
}

/**
 * Pauses the MIDlet; this method does nothing because there
 * are no background activities or shared resources to close.
 */
public void pauseApp() {
}

/**
 * Destroys the MIDlet; this method does nothing because
 * there is nothing to cleanup that is not handled by the
 * garbage collector.
 */
public void destroyApp(boolean unconditional) {
}

/*
 * Responds to the user's selection of abstract commands.
 * This MIDlet has only an exit command. This method
 * responds to it by cleaning up and notifying the system
 * that the MIDlet has been destroyed.
 */
public void commandAction(Command command, Displayable screen) {
    if (command == exitCommand) {
        destroyApp(false);
        notifyDestroyed();
    }
}
}

```


Index

A

- About boxes, 2
- abstract commands, 4, 9, 11
 - associating with screens, 10
 - associating with screens, example, 12
 - handling, 12
 - instantiating, 10
 - instantiating, example, 12
 - labels, 11
 - priorities, 11
 - types, 11
- acquiring shared resources, 13
- active state, 5
- adding certificates to JAD files, 26, 27
- APIs, security-sensitive, 15
- architecture, J2ME, 2
- assigning listeners, 10
- associating screens and abstract commands, 10
 - example, 12
- attributes
 - JAD file, 25
 - JAR file, 23
 - MicroEdition-Profile, 24
 - MIDlet-Certificate-*m-n*, 26
 - MicroEdition-Configuration, 24
 - MIDlet-Jar-RSA-SHA1, 27
 - MIDlet-Jar-Size, 25
 - MIDlet-Jar-URL, 25
 - MIDlet-*n*, 24
 - MIDlet-Name, 23, 25
 - MIDlet-Permissions, 23
 - MIDlet-Permissions-Opt, 23
 - MIDlet-Vendor, 23, 25
 - MIDlet-Version, 23, 25

B

- bug parade
 - category, 32
 - URL, 32
- bug report
 - information in, 33
 - URL, 33
- bugs
 - reporting, 32 to 33

C

- canvases, 3
- certificate, 26
- certificate authority, 26
- certificate chain, 26, 27
- certificates
 - adding to JAD files, 26, 27
 - listing, 27
- CLDC, 1
 - debugging architecture, 31
 - in J2ME platform architecture, 2
 - prohibited features in, 19, 20
- commandAction method, 12
 - example, 13, 36
- CommandListener interface, 10, 12
- commands, abstract, 9
- compiler, javac, 18
- compiling
 - directory sequence for preverifying and, 17
 - MIDlets, 17 to 20
 - MIDlets that use push functionality, 18
- constraints
 - text box, 11
 - TextField.ANY, 11

- constructors
 - example of MIDlet, 11
 - MIDlet, 10

- creating
 - JAD files, 25
 - JAR files, 22
- creating MIDlets, 9 to 15

D

- debugging a MIDlet, 31 to 32
- destroyApp method, 13
 - example, 14, 36
- destroyed state, 5
- device
 - root CA and, 27
 - security policy, 6
- directories
 - preverifier default, 19
 - sequence for compiling and preverifying, 17

E

- environment, screen-based, 2
- establishing trust, 6, 26
- examples
 - abstract commands, 11
 - commandAction method, 13, 36
 - destroyApp method, 14, 36
 - Hello MIDlet, 35 to 36
 - HTML file, 29
 - instantiating screens, 12
 - JAD file, 25
 - JadTool, 26, 27
 - JAR file creation, 24
 - life cycle methods, 14
 - listener, 12
 - manifest file, 24
 - MIDlet constructor, 11, 36
 - midp command, 31, 32
 - pauseApp method, 14, 36
 - startApp method, 14, 36
 - text box, 11

F

- feedback address
 - midp-comments@sun.com, 32

G

- get-hello.html file, 29

H

- handling
 - abstract commands, 12
 - user input, 12
- HelloMIDlet.jad file, 25
- HelloMIDlet.mf file, 24
- HTML files
 - example, 29
 - link to JAD file, 29

I

- initial screen, setting, 10
- instantiating
 - abstract commands, 10
 - abstract commands, example, 12
 - screens, 10
 - screens, example, 12
- intermediate certificates, 27

J

- J2ME architecture, 2
- J2SE keystore, 26
- JAD files, 21
 - adding certificates to, 26, 27
 - attributes of, 25
 - creating, 25
 - example, 25
 - link to in HTML file, 29
 - listing certificates in, 27
 - MIME type of, 29
- JadTool utility, 26
 - example of, 26, 27
- jar command, 22, 24
- JAR file, 19
- JAR files, 21
 - attributes of, 23
 - contents of, 22
 - creating, 22
 - example of creating, 24
 - MIME type of, 29
 - signing, 26 to 27
 - signing, example of, 27

- Java Platform Debugger Architecture, 32
 - Code Warrior, 32
 - Forte for Java, 32
 - JBuilder, 32
 - jdb, 32
 - Sun ONE Studio, 32
 - Visual Cafe, 32
- Java Product Versioning Specification, 23
- javac, 18, 31
- javax.microedition.io.Connector
 - class, 15, 23
- javax.microedition.io.Connector.comm
 - permission name, 7, 23
- javax.microedition.io.Connector.
 - datagram permission name, 7, 23
- javax.microedition.io.Connector.
 - datagramreceiver permission name, 7, 23
- javax.microedition.io.Connector.http
 - permission name, 7, 23
- javax.microedition.io.Connector.https
 - permission name, 7, 23
- javax.microedition.io.Connector.
 - serversocket permission name, 7, 23
- javax.microedition.io.Connector.
 - socket permission name, 7, 23
- javax.microedition.io.Connector.ssl
 - permission name, 7, 23
- javax.microedition.io.HttpConnection
 - class, 15, 23
- javax.microedition.io.HttpsConnection
 - class, 15, 23
- javax.microedition.io.PushRegistry
 - class, 15, 23
- javax.microedition.io.PushRegistry
 - permission name, 7, 23
- javax.microedition.lcdui package, 10, 15
- javax.microedition.lcdui.game
 - package, 15
- javax.microedition.media package, 15
- javax.microedition.media.controls
 - package, 15
- javax.microedition.midlet package, 10, 15
- javax.microedition.rms package, 15

K

- keytool utility, 26
- KVM debug proxy, 32

L

- labels, abstract commands, 11
- life cycle, 5
 - methods, 13 to 14
- linking to JAD files from HTML files, 29
- listeners, 10, 12
 - assigning, 10
 - associating with screens, example, 12
- listing certificates in JAD files, 27

M

- manifest files
 - attributes in, 23
 - creating, 23
 - example, 24
- MicroEdition-Configuration attribute, 24
- MicroEdition-Profile attribute, 24
- MIDlet class, 10
- MIDlet constructors, 10
 - example, 11
- MIDlet environment, 2 to 4
- MIDlet life cycle, 5
- MIDlet suites, 1
 - components of, 21
 - creating, 21 to 27
 - establishing trust, 6, 26
 - in J2ME platform architecture, 2
 - JAD file attributes, 25
 - JAR file attributes, 23
 - publishing, 29 to 30
 - resource of, 22
 - resources, 21
 - signature of, 26
 - signing, 6
 - trusted, 6
 - untrusted, 6
- MIDlet-Certificate-*m-n* attribute, 26
- MIDlet-Jar-RSA-SHA1 attribute, 27
- MIDlet-Jar-Size attribute, 25
- MIDlet-Jar-URL attribute, 25
- MIDlet-*n* attribute, 24
- MIDlet-Name attribute, 23, 25
- MIDlet-Permissions attribute, 23
- MIDlet-Permissions-Opt attribute, 23

- MIDlets, 1
 - compiling, 17 to 20
 - constructor example, 36
 - constructors, 10
 - creating, 9 to 15
 - debugging, 31 to 32
 - getting, 6
 - in J2ME platform architecture, 2
 - JAD file attributes, 25
 - JAR file attributes, 23
 - packaging, 21 to 27
 - preverifying, 17 to 20
 - publishing, 29 to 30
 - push functionality and compiling, 18
 - running, 6
 - screen-based environment, 2
 - states and state transitions, 5
 - untrusted, 26
- MIDletStateChangeException, 13
- MIDlet-Vendor attribute, 23, 25
- MIDlet-Version attribute, 23, 25
- MIDP, 1
 - in J2ME platform architecture, 2
 - specification, 1
- midp command, 31
 - example, 31, 32
- midp-comments@sun.com feedback address, 32
- MIDs, 1
- MIME types, 29
- modifiers
 - text box, 11
 - TextField.UNEDITABLE, 11

O

- output directory of preverify command, 19
- over the air provisioning, 29

P

- packages
 - javax.microedition.lcdui, 10
 - javax.microedition.midlet, 10
- packaging MIDlets, 21 to 27
- pauseApp method, 13
 - example, 14, 36
- paused state, 5

- permission names

- javax.microedition.io.Connector.
 - comm, 7
 - javax.microedition.io.Connector.
 - datagram, 7
 - javax.microedition.io.Connector.
 - datagramreceiver, 7
 - javax.microedition.io.Connector.
 - http, 7
 - javax.microedition.io.Connector.
 - https, 7
 - javax.microedition.io.Connector.
 - serversocket, 7
 - javax.microedition.io.Connector.
 - socket, 7
 - javax.microedition.io.Connector.
 - ssl, 7
 - javax.microedition.io.
 - PushRegistry, 7

- permissions, 7

- preverify command, 19

- preverifying

- directory sequence for compiling and, 17
 - MIDlets, 17 to 20
 - prohibited CLDC features and, 20

- priorities, abstract commands, 11

- problems

- reporting, 32 to 33

- publishing MIDlet suites, 29 to 30

- push functionality, compiling MIDlets that use, 18

R

- releasing shared resources, 13

- reporting problems, 32 to 33

- root CA, 27

- RSA key pair, 26

S

- saving persistent state, 13

- screen

- listeners, assigning, 10

- screen-based environment, 2

- screens
 - associating with abstract commands, 10
 - associating with abstract commands, example, 12
 - associating with listener, example, 12
 - canvases, 3
 - instantiating, 10
 - instantiating, example, 12
 - listeners, 10, 12
 - setting initial, 13
 - structured, 3
 - text box, 9, 11
 - unstructured, 3
- security
 - interaction modes, 7
 - policies, 6
- security-sensitive APIs, 6, 15, 26
 - `javax.microedition.io.Connector`, 15
 - `javax.microedition.io.HttpConnection`, 15
 - `javax.microedition.io.HttpsConnection`, 15
 - `javax.microedition.io.PushRegistry`, 15
 - requests to use, 23
- setting initial screens, 10, 13
- shared resources
 - acquiring, 13
 - releasing, 13
- signing JAR files, 26 to 27
 - example, 27
- specification, MIDP, 1
- `startApp` method, 13
 - example, 14, 36
- structured screens, 3
 - text box, 9, 11
- system menus, 4

T

- text boxes, 9, 11
 - constraints, 11
 - modifiers, 11
 - `TextField.ANY` constraint, 11
 - `TextField.UNEDITABLE` modifier, 11
- trusted MIDlet suites, 6
- trusted MIDlets
 - MIDlets trusted, 26
- types, abstract commands, 11

U

- unstructured screens, 3
 - canvases, 3
- untrusted MIDlet suites, 6
- untrusted MIDlets, 26
- URLs
 - bug parade, 32
 - for bug reports, 33
 - MIDP specification, 1

