Please Recycle

Adobe PostScript

Contents

Figures

Code Samples

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 `.login` file. Use `ls -a` to list all files. `% You have mail.`
`AaBbCc123`	What you type, when contrasted with on-screen computer output	`% su` `Password:`
AaBbCc123	Book titles, new words or terms, words to be emphasized	Read Chapter 6 in the *User's Guide*. These are called *class* options. You *must* be superuser to do this.
	Command-line variable; replace with a real name or value	To delete a file, type `rm` *filename*.

Shell Prompts

Shell	Prompt
C shell	`%`
Microsoft Windows	*directory*>

Related Documentation

The following documentation is included with this release:

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

Accessing Sun Documentation Online

The Java Developer Connection[sm] web site enables you to access Java platform technical documentation on the Web:

http://dev /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:

CHAPTER 1

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

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:

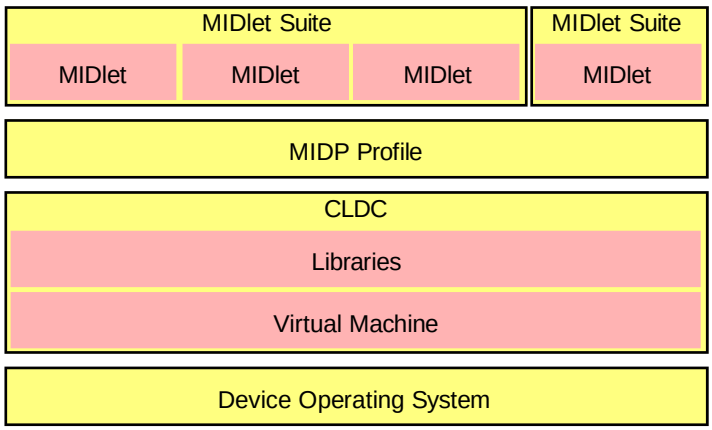

FIGURE 1 J2ME Platform Architecture for MIDP and MIDP Applications

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 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:

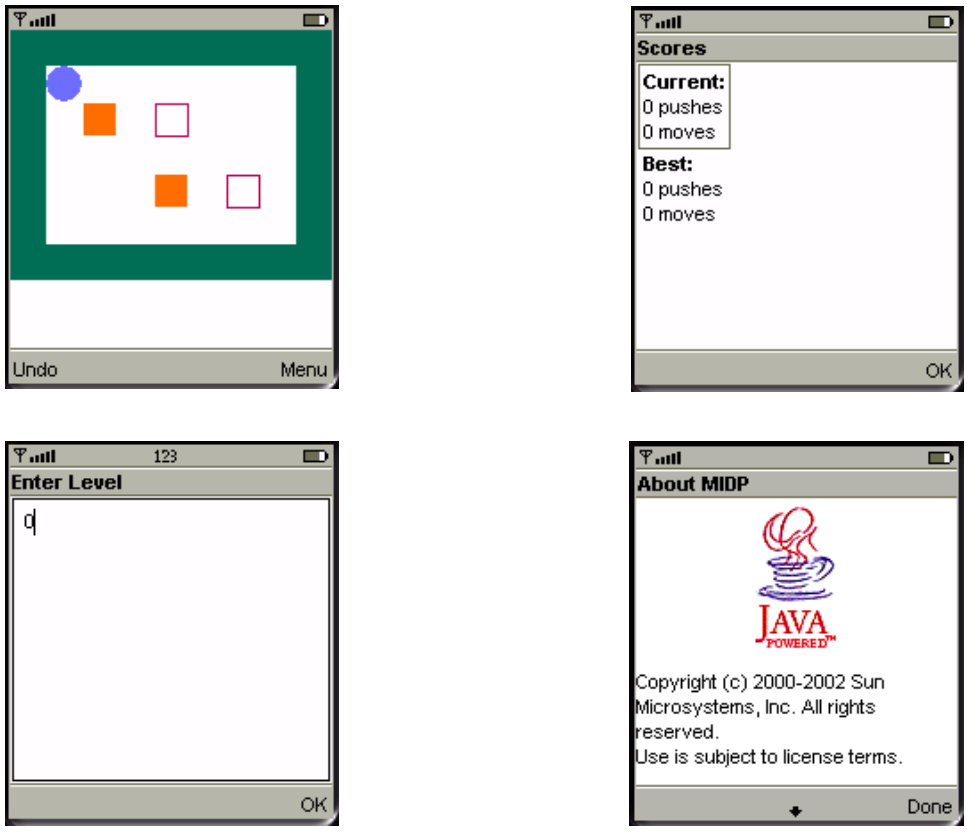

FIGURE 2 Screens of the PushPuzzle Game

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 *command*. A screen can have any number of commands; each screen should have at least one. commands enable you to define actions without specifying their user interface. MIDP implementations determine their presentation. This makes commands portable, like structured screens. MIDlets that use them can look and behave appropriately on different devices without code changes.

When presenting 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 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.

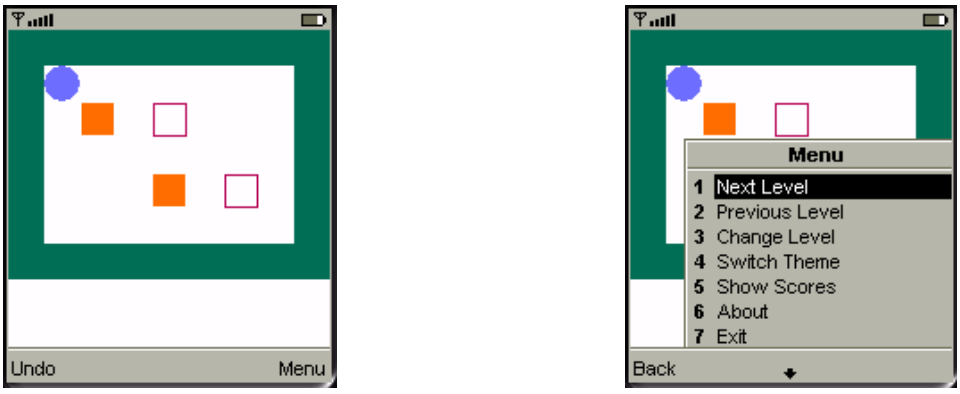

FIGURE 3 Screen and Associated System Menu of Commands

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 commands, that enable a user to carry out their tasks when they run the MIDlet.