
Simplicity for Palm OS Platform

User Guide

Simplicity for Palm OS Platform

Simplicity for Palm OS Platform 1.2.2 by Data Representations, Inc.

Manual Version 1.2.2

Copyright © 1997 - 2000 Data Representations, Inc.

All rights reserved.

Copyright © 1997 - 2000 Data Representations, Inc. All rights reserved. This software and documentation are copyrighted. All rights, including ownership of the software, are reserved to Data Representations, Inc.

Simplicity Professional contains InstallShield installation software from InstallShield Software Corporation. (<http://www.installshield.com/>) InstallShield is a trademark of InstallShield Software Corporation.

Data Representations, the Data Representations logo, Simplicity for, Simplicity Professional, Execution-on-the-fly, and Code Sourcerer are trademarks of Data Representations, Inc. Solaris, Java and JavaBeans are trademarks of Sun Microsystems, Inc. Pentium is a trademark of Intel Corporation. IBM, OS/2, OS/2 Warp and AIX are trademarks of International Business Machines Corporation. Microsoft, Windows, Windows NT, and Windows 95 are trademarks of Microsoft Corporation. Apple, Macintosh and MacOS are trademarks of Apple Computer, Inc. InstallShield is a trademark of InstallShield Software Corporation. Adobe Acrobat is a trademark of Adobe, Inc. Palm OS Platform is a trademark of Palm, Inc. All other brand and product names are trademarks of their respective owners.

Table of Contents

Simplicity for Palm OS Platform

Table of Contents

CHAPTER 1	<i>Installation</i>	1
System Requirements		
<i>Java Virtual Machine</i>		1
<i>Minimum System Requirements</i>		2
<i>Suggested Minimum System Features</i>		2
Installing		
<i>Windows (95/98/NT)</i>		2
<i>Mac OS</i>		2
<i>OS/2 Warp</i>		3
<i>Linux, Solaris, AIX, IRIX, SCO UnixWare, and Other Unix</i>		3
Starting Simplicity for Palm OS Platform		
<i>ClassPath</i>		4
<i>Starting Simplicity for Palm OS Platform</i>		4
Platform Specific Notes		
<i>OS/2 Warp</i>		4
<i>Linux</i>		5
<i>Macintosh</i>		5
<i>Solaris</i>		5
<i>SCO UnixWare</i>		5
The Personal Settings Directory		
<i>Linux, Solaris, SCO UnixWare, and other Unix</i>		6
<i>Windows, MacOS, and OS/2</i>		6
<i>Files in the Personal Settings Directory</i>		6
Technical Support and Feedback		
<i>Technical Support</i>		7
<i>Feedback</i>		7

CHAPTER 2	<i>Tutorial 1 - Introduction to Simplicity for the Palm OS Platform</i>	9
------------------	---	----------

A Simple Tip Calculator

<i>Open a project.....</i>	<i>10</i>
<i>Create a new Application.....</i>	<i>10</i>
<i>Assemble the GUI.....</i>	<i>11</i>
<i>Cleaning up.....</i>	<i>13</i>
<i>Responding to events</i>	<i>14</i>
<i>Creating an Exit button.....</i>	<i>16</i>
<i>Compiling the program.....</i>	<i>17</i>
<i>Packaging your application.....</i>	<i>18</i>

CHAPTER 3	<i>Integrated Design Environment</i>	21
------------------	--------------------------------------	-----------

The IDE Window

<i>The Classpath</i>	<i>23</i>
<i>The Folders area.....</i>	<i>23</i>
<i>Using the Classpath and the Folders area</i>	<i>23</i>
<i>Project Groups.....</i>	<i>23</i>

Editing parts of the Project Tree

<i>Editing Groups using the IDE Group Editor</i>	<i>24</i>
<i>Editing The Classpath.....</i>	<i>25</i>
<i>Editing The Folders area.....</i>	<i>25</i>
<i>Opening items in the Group Contents Box</i>	<i>26</i>

IDE Menu Bar

<i>IDE Button Bar</i>	<i>27</i>
<i>File Menu.....</i>	<i>27</i>
<i>Edit Menu.....</i>	<i>29</i>
<i>Create Menu.....</i>	<i>29</i>
<i>Import Menu</i>	<i>30</i>
<i>Project Menu.....</i>	<i>30</i>
<i>Help Menu.....</i>	<i>30</i>

Program Settings

<i>Directories</i>	<i>31</i>
<i>External Editors.....</i>	<i>32</i>

<i>Object Palette</i>	<i>32</i>
<i>Java Editor.....</i>	<i>33</i>
<i>Printing.....</i>	<i>33</i>

CHAPTER 4	<i>Java Source Code Editor</i>	35
------------------	--------------------------------	-----------

Editing

<i>File Menu.....</i>	<i>36</i>
<i>Edit Menu.....</i>	<i>36</i>
<i>Indentation Features.....</i>	<i>37</i>
<i>Color and Printing Features.....</i>	<i>38</i>
<i>Search & Replace</i>	<i>38</i>
<i>The Sourcerer's Apprentice.....</i>	<i>39</i>

CHAPTER 5	<i>Bitmap Editor</i>	41
------------------	----------------------	-----------

Using bitmaps within a Spotlet

Using the Bitmap Editor

<i>Tools.....</i>	<i>42</i>
-------------------	-----------

CHAPTER 6	<i>The Spotlet Composer</i>	45
------------------	-----------------------------	-----------

Creating a New Composer

<i>The Spotlet Composer.....</i>	<i>46</i>
----------------------------------	-----------

The Composer Window

<i>Composer Button Bar</i>	<i>47</i>
<i>File Menu.....</i>	<i>48</i>
<i>Program Menu.....</i>	<i>48</i>
<i>Code Menu.....</i>	<i>48</i>
<i>Parts Menu.....</i>	<i>49</i>

Property Notebooks

<i>Notebook pages.....</i>	<i>49</i>
<i>Spotlet Properties.....</i>	<i>50</i>
<i>Spotlet code areas</i>	<i>51</i>

Code and Application Generation

<i>Compile against the J2ME.....</i>	<i>52</i>
<i>Generate a Jar file</i>	<i>53</i>
<i>Convert to PRC.....</i>	<i>53</i>
<i>Load classes onto the Palm OS Platform</i>	<i>54</i>

CHAPTER 7 *Object Palette* **55**

Assembling A Program Using The Object Palette

<i>Object Palette</i>	<i>56</i>
<i>Palm OS Platform Emulator.....</i>	<i>56</i>
<i>Object Palette Pages.....</i>	<i>56</i>

Kjava Parts

<i>CheckBox</i>	<i>57</i>
<i>RadioButton</i>	<i>57</i>
<i>Button.....</i>	<i>58</i>
<i>TextField</i>	<i>58</i>
<i>TextBox.....</i>	<i>58</i>
<i>Scroll Text Box</i>	<i>58</i>
<i>Select Scroll Text Box.....</i>	<i>59</i>
<i>Slider.....</i>	<i>59</i>
<i>ValueSelector</i>	<i>59</i>

The Palm OS Platform Emulator

<i>Emulator Options.....</i>	<i>61</i>
<i>Building Spotlets</i>	<i>62</i>

CHAPTER 8 *Code Sourcerer* **63**

Using the Code Sourcerer

<i>Change a property of an existing part</i>	<i>65</i>
<i>Ask a part about one of its properties</i>	<i>65</i>
<i>Palm OS Platform database operations</i>	<i>65</i>
<i>Palm OS Platform Beaming operations.....</i>	<i>66</i>
<i>Palm OS Platform Screen Painting</i>	<i>67</i>
<i>Palm OS Platform Network operations</i>	<i>67</i>
<i>Palm OS Platform Spotlet operations.....</i>	<i>68</i>
<i>Perform floating point computations.....</i>	<i>68</i>
<i>Declare a new variable.....</i>	<i>69</i>
<i>Java system operations</i>	<i>69</i>

<i>Miscellaneous</i>	<i>70</i>
<i>Java Language statements</i>	<i>70</i>

CHAPTER 9	<i>Java Command Window</i>	73
------------------	----------------------------	-----------

Using The Java Command Window	
<i>Command Input.....</i>	<i>74</i>
<i>Local Symbol Table.....</i>	<i>75</i>
<i>Command History.....</i>	<i>75</i>

The Three Java Command Window Contexts	
<i>IDE.....</i>	<i>75</i>
<i>Composer.....</i>	<i>76</i>
<i>Debugger</i>	<i>76</i>

CHAPTER 10	<i>Debugger</i>	77
-------------------	-----------------	-----------

Starting the Debugger

The Debugger Window	
<i>Available classes and methods.....</i>	<i>79</i>
<i>Breakpoints</i>	<i>79</i>
<i>Threads</i>	<i>79</i>
<i>Execution Stack.....</i>	<i>80</i>
<i>Source Viewer</i>	<i>80</i>
<i>Variables List</i>	<i>80</i>
<i>Command Buttons.....</i>	<i>80</i>
<i>Java Command Window.....</i>	<i>81</i>
<i>Run program</i>	<i>81</i>
<i>Load classes.....</i>	<i>81</i>

CHAPTER 11	<i>Advanced Features - Extending the IDE</i>	83
-------------------	--	-----------

Extending the IDE	
<i>The IDEmenu.config file.....</i>	<i>83</i>
<i>MenuBar</i>	<i>84</i>
<i>Menu</i>	<i>84</i>
<i>MenuItem</i>	<i>84</i>

<i>Separator</i>	<i>85</i>
<i>Action</i>	<i>85</i>

Samples of the extended IDE

<i>Adding a command to the help menu.....</i>	<i>86</i>
<i>Adding a new action</i>	<i>87</i>
<i>Modifying existing actions</i>	<i>87</i>
<i>A Complex Action</i>	<i>88</i>

Index

This chapter will discuss installing Simplicity for Palm OS Platform.

It will cover

- System requirements
- The install program
- Operating system specific notes

System Requirements

Java Virtual Machine

Before you install Simplicity for Palm OS Platform, you must install a Java Development Kit (JDK). If you do not have a JDK already installed you can get one from

<http://www.javasoft.com/cgi-bin/java-ports.cgi>

This web page lists all of the known JDKs by operating system. Make sure that you choose one which is compliant with Java version 1.3 or later. Try to use the latest revision of the JDK for your operating system.

Minimum System Requirements

- Java Development Kit, version 1.3 or higher
- 10 MB free disk space for program
- HTML web browser

Suggested Minimum System Features

- CPU of similar power to a 133 MHz Pentium chip
- 32 MB RAM (64 MB on MacOS)
- 800x600 pixel display

Installing

NOTE: Before beginning the install, you must have installed a JDK1.3 or greater on your computer. This must be a full JDK; a JRE is not enough.

Windows (95/98/NT)

The installer is called `simpalmp.exe`. Double click this file to begin the install. The install utility will guide you through the installation. You must accept the terms of the licence agreement to complete the install. After specifying the install location, all of the files will be copied.

Mac OS

Make sure that you have previously installed the MRJ 2.1.4, available from ftp://ftp.apple.com/developer/Java/MRJ_2.1.4_Web_Install.sit.hqx. MRJ 2.2 is not recommended at the current time.

See the platform specific notes on page 5 before installing on the Mac OS.

The installer is MacBinary encoded and should be automatically decoded when downloaded. If not, you can decode it using Stuffit Expander 4.5 or later. Double click the installer to begin the install.

OS/2 Warp

The distribution is called `simpalmp.zip`. Unzip this file using an unzip utility, being careful to maintain the directory structure. Execute the `simpwps.cmd` Rexx script to create a Program Object on your desktop for starting Simplicity.

Linux, Solaris, AIX, IRIX, SCO UnixWare, and Other Unix

Simplicity is distributed as a `.tar.gz` file, named `simpalmp.tar.gz`. To install, copy the distribution to your user directory and execute the following commands to unpack the archive:

```
gzip -d simpalmp.tar.gz
tar -xf simpalmp.tar
```

Starting Simplicity for Palm OS Platform

On most platforms, the installer will create a launcher which will start Simplicity for Palm OS Platform.

- **Windows 95, 98 or NT** A group called ‘Simplicity for Palm OS Platform’ will be created in your Programs menu on the Start Menu.
- **OS/2 Warp** There is also a file called `simpwps.cmd` which will create a Program Object on your desktop from which you can start Simplicity for Palm OS Platform.
- **Solaris, Linux, and most other Unix systems** a shell script will be created in the directory where Simplicity is installed, called `SimplicityPalm`. You must edit the two variables in this file which correspond to the location of the java executable on your machine and the location of the Simplicity install directory. Set this script to be executable (using the `chmod` command), and then you can copy this file to a directory on your path.
- **Macintosh** A program icon called “Simplicity for Palm OS Platform” will be created in the install directory. Double-click this icon to start Simplicity. You may wish to make an alias of this file and put it on the desktop.

In many cases the default installation is sufficient. There are cases when you may need to customize the installation. The following information is provided to help you customize the Simplicity for Palm OS Platform installation.

ClassPath

Simplicity requires that one item be on your classpath. It is

1. The full path of `simplicitypos.jar`

It is found in the Simplicity install directory. The method by which you set your system classpath varies according to the operating system.

- **Windows 95** The CLASSPATH is set in your `autoexec.bat` file with
`SET CLASSPATH=.;c:\SimplicityPalmOS\simplicitypos.jar;`
- **Windows NT** The CLASSPATH is set in the Environment page of the System Properties in the Control Panel. Set a system variable similar to the Windows 95 example, above.
- **OS/2 Warp** The CLASSPATH is set in your `config.sys` file with a line similar to the Windows 95 example, above.
- **Solaris/Linux/other Unix** The CLASSPATH is set in your `.profile` with
`export CLASSPATH =${CLASSPATH}:
 $HOME/SimplicityPalmOS/simplicitypos.jar:`

Note that the CLASSPATH entries should be on one line; some have been broken up here for clarity.

Starting Simplicity for Palm OS Platform

Simplicity for Palm OS Platform is started by invoking the Java interpreter. The main entry point is `'datarep.Simplicity'`. On most systems, the following command can be used to start Simplicity.

```
java -mx100m datarep.Simplicity
```

Platform Specific Notes

OS/2 Warp

The installation directory for Simplicity for Palm OS Platform as well as the Project directory must be located on an HPFS drive.

OS/2 users may wish to execute the `simpwps.cmd` file. This REXX script will create a WPS object on the OS/2 desktop for starting Simplicity.

We recommend using `jdk13` from IBM. IBM's Software Choice page is located at <http://www.ibm.com/software/os/warp/swchoice/>.

The debugger needs to have the localhost interface enabled. You can do this by typing the following command in an OS/2 window:

```
ifconfig lo 127.0.0.1 up
```

Note that this will be reset when you next reboot your computer.

Linux

Before installing Simplicity, make sure that you have the latest Linux JDK from www.blackdown.org, Sun, or IBM.

Macintosh

There is not currently a full version of JDK 1.3 available for the Macintosh. While most of the functions of Simplicity for Palm OS Platform will work on the Macintosh, it is possible that projects created on other machines will not work on the Macintosh.

It is highly recommended that you have at least 64 MB of RAM to run Simplicity for Palm OS Platform. MRJ 2.1.4 is the recommended version of the MRJ.

Solaris

On some Solaris machines, Simplicity for Palm OS Platform will stall shortly after starting. This can be fixed by disabling the JIT. (Uncomment the appropriate line in the startup script.) Solaris 8 users will need to be sure to edit the first line of the shell script as described in the script.

SCO UnixWare

On some SCO machines, Simplicity for Palm OS Platform will stall shortly after starting. This can be fixed by removing the `-mx100m` argument from the startup script. (Uncomment the appropriate line in the startup script.)

The Personal Settings Directory

Simplicity has a centralized directory where all of the settings files for Simplicity are stored. The name for the directory is `‘.simplicitypalm’`. The default location for the directory is different in Unix (including Linux) and in other operating systems.

Linux, Solaris, SCO UnixWare, and other Unix

In Unix, the default location for the settings directory is in the user’s home directory. For example, a user named bob with a home directory `/home/bob`, would have the default settings directory located in `/home/bob/.simplicitypalm`. Multiple users in Unix can run Simplicity from the same location, but have their own unique settings. In Unix, the `‘.’` which precedes the directory name marks the directory as a settings directory, and as such is not normally displayed by the `ls` command. However, `ls -a` will list all files, including ones which start with `‘.’`.

Windows, MacOS, and OS/2

Windows, Macintosh, and OS/2 are primarily single-user operating systems. As such, the personal settings directory on these platforms is located in the directory where Simplicity is installed.

Files in the Personal Settings Directory

There are several files and directories which will have special meaning to Simplicity if found in the Personal Settings Directory.

- ***Project files*** Every project stores its settings in a file in the Personal Settings Directory with a name constructed by adding `.Simplicity` to the end of the project name. (i.e. `Tutorial1.Simplicity`).
- ***program.set*** This file contains the configuration settings for Simplicity, including window locations and printing preferences.
- ***IDEmenu.config*** This file is used to customize the IDE (see page 83).
- ***.lic_txt*** and the contents of the ***.license*** directory are files generated by Simplicity for Palm OS Platform and used to store license information. Users should not change these files.
- ***PalmBeamClipboard*** This file is used to simulate a beam being sent from or to the Emulator. Users may wish to edit this file manually, or by using the Beam Clipboard Editor (see page 60).

- ***PalmDatabase*** This directory is used by the Emulator to simulate the database present on Palm OS Platform based handheld computers.

Technical Support and Feedback

Technical Support

Send your questions to: support@datarepresentations.com

We ask that you restrict your questions to issues specific to Simplicity for Palm OS Platform.

Be sure to include your name and return email address. Please also indicate which Operating System and Java Virtual Machine you are using.

Feedback

We always love to hear from our customers. Please send us your comments.

We are constantly improving our products. Please send us your ideas and suggestions for how we can improve Simplicity for Palm OS Platform.

Send feedback to: info@datarepresentations.com

Tutorial 1 - Introduction to Simplicity for the Palm OS Platform

Welcome to Simplicity for Palm OS Platform. This tutorial will guide you through creating a simple tip calculator using Simplicity for Palm OS Platform.

By completing this tutorial you will learn about

- Managing files in the IDE
- Building a Graphical User Interface (GUI) for a program which can run on the Palm OS Platform
- Creating event code using the Code Sourcerer
- Packaging a program and placing it on a Palm OS Platform based handheld computer

It is estimated that this tutorial should take most people between 30 and 40 minutes to complete. All of the details in this tutorial are designed to highlight important functionality, so try to read and follow each part carefully.

A Simple Tip Calculator

If you haven't already done so, start Simplicity for Palm OS Platform in the manner appropriate for your operating system. The Simplicity IDE should be on the screen.

Open a project

You will now open a project which came with Simplicity for Palm OS Platform called Tutorial1.

1. Choose Open from the File menu.
If the welcome screen is showing, you can choose ‘Open an Existing Project’ by clicking the mouse on the green disk icon. Otherwise, you can choose ‘Open Project’ from the File menu or click the ‘Open’ button (green disk icon) on the button bar.
2. A small window will appear listing the names of all the available projects. Choose ‘Tutorial1’ from the list and press the Ok button.

Create a new Application

You will now create a new Spotlet (which will become your tip calculator). A Spotlet is a Java application which can run on the Palm OS Platform. In the IDE, you see on the left the Project Tree, which has three sections. The first is called “Project Groups”, and contains a list of group names. Each of these items represents a group of files which will appear in the box on the right when selected. For example, if you chose ‘Java Source Files’, all of the Java source files would appear in the box on the right.

1. Choose the first item in Project Groups, labeled ‘Composer Files’.
2. To see what our tutorial will look like when finished, double click on Finished.Spotlet.

Three windows appear. One of these will contain a picture of a Palm OS Platform based handheld computer. Running in it will be the simple tip calculator. It will have two spaces where you can enter text, labeled ‘Bill Amount \$:’ and ‘Percentage:’. It will also have a button labeled ‘compute tip’. Click to the right of ‘Bill Amount \$:’ so that the cursor starts to flash. You can then type a number into the TextField. Click on ‘Percentage’ and type another number. Finally, press the ‘compute tip’ button to see the amount of the tip.

3. When you are finished, choose Exit from the File menu of the first window, titled ‘Simplicity Composer’. Press **No** when asked if you would like to save any changes.
4. In the IDE, choose ‘*Spotlet*’ from the **Create** menu. This creates a new Spotlet in the current project. (You can also press the ‘Create a Spotlet’ button on the button bar. It is the fourth button from the left in the IDE.)

A new Icon appears in the ‘Composer Files’ group with a name similar to Untitled0.Spotlet.

5. Click this new icon once to select it. (It will become highlighted.)
6. From the **Edit** menu, choose **Rename selection**. A text field will appear to the right of the icon. Replace the old name with ‘tip calculator’. Click the mouse anywhere else in the right hand window to accept the change. If you don’t specify the ‘.Spotlet’ extension, Simplicity will add it for you. Simplicity will also modify the spacing and capitalization to conform to Java naming conventions.
7. Double click the new ‘TipCalculator.Spotlet’ icon to open the Simplicity Composer.

You have now created a new Spotlet, and have loaded it into the Simplicity Composer. You should see three new Windows on the screen. They are the Composer window, the Object Palette, and a window which contains a Palm OS Platform based handheld computer. This last window is the Palm OS Platform Emulator.

The composer window is the place where you will set up all of the properties of the graphical parts of your program. It is also where you will be able to supply all the code that your program will need.

Toward the top of the Composer window is a choice box. This is the Part List. If you select it, you will see that there is currently only one item in it called TipCalculator. This refers to the Spotlet that will become your application.

The bottom portion of the Composer window has a seven page notebook. The first page, called ‘Spotlet’, has the properties that are specific to a Spotlet application running on the Palm OS Platform. There is also a checkbox which enables you to edit the location of parts in your GUI using your mouse.

1. Click the ‘Edit locations using mouse’ checkbox so that it is checked. This will make it easier to set up your GUI.

Assemble the GUI

You will now begin to assemble the graphical components for your tip calculator. All of the graphical components can be found in the Object Palette window.

1. On the Object Palette, make sure that the first page, ‘Kjava’, is visible. (If not, click the mouse button on the ‘Kjava’ Tab.)

2. You will see icons for all of the available parts. Click the fourth icon, named 'TextBox', with the mouse. (It should become depressed. Also, the title bar of the Object Palette will reflect your selection.)
3. In the Emulator window, click once near the top of the screen. You have just placed a TextBox into your program. You should see the word 'textBox1' surrounded by a red squares. In the Composer window, a new page should have appeared, which displays the TextBox properties.
4. If the TextBox is not already centered at the top of the Emulator screen, you can center it by clicking inside the red square and dragging. Clicking on the red squares on the sides and corners of the box will resize the box. At this stage, try not to resize the part (but if you change the size a little it is OK).
5. We will now place the TextFields for our application in the Emulator. Click the 'TextField' icon in the Object Palette.
6. In the Emulator window, click once on the left side of the Emulator, below the TextBox. You should see a new part, which will have the word "textField2:" inside a red square.
7. Click the 'TextField' icon in the Object Palette again to get ready to place another TextField.
8. In the Emulator window, click once below textField2. You should see a new part in the Emulator, named 'textField3'. Move this TextField so it is lined up with the first.
9. Next you will create the button which will compute the tip. Click the first icon in the Object Palette, named 'Button'.
10. In the Emulator window, click once beneath textField3. You should see the button, named 'button4'.
11. Finally you will create the box which will show the results of the calculation. Click the fourth icon in the Object Palette, titled 'TextBox' and place it beneath the button near the bottom of the screen. (If part of the red square is extending off the bottom of the screen, it is OK for now because you will be changing the size of the TextBox in the next section.)

If by accident you placed a part you did not mean to place, you can remove it by using the 'Recycle' button. Choose the part by clicking it with the mouse button. Then push the 'Recycle Current Part' button located toward the top of the Composer window. The part will be removed from your program and will appear as an icon in the 'Recycled' page in the Object palette. You can select it from the palette if you wish to reuse it.

Cleaning up

You have finished placing all of the components that you will need for the simple tip calculator. You will now go through your GUI and clean up the components you created. First you will set up the title of your application.

1. Click on the TextBox titled 'textBox1' in the Emulator. You will see the Properties page for this TextBox appear in the Composer.
2. Change the **TextBox text** from 'textBox1' to 'Tip Calculator' by typing in the text area of the Composer window. Note that the text changes immediately in the Emulator as you type. If you did not change the size of the TextBox, the TextBox will wrap the text to two lines because it is too long to fit onto one.
3. You can change the height and width of the TextBox by editing the numbers in the **Size** box on the Properties page. Set the height to 15, and the width of the TextBox to 80. This should be big enough to put all of the text on one line. When you change the width of the TextBox, a warning symbol may appear in the upper left hand corner of the Emulator window. Pressing this symbol will cause the warning associated with it to be displayed in the Java Console.
4. Next you will set up the TextField where you can enter the amount of the bill. You can switch to the property sheet for this part by clicking 'textField2' in the Emulator. The property notebook for textField2 should appear in the Composer.
5. Change the **TextField label** from 'textField2' to 'Bill Amount \$'. Note that you do not have to type the ':' which appears in the Emulator; that is added automatically by the TextField.
6. Each component is given a name which you use to refer to each component when you write Java code. The current **Object name** is set to 'textField2'. Change it to a simpler, more descriptive name, 'amount'.
7. To edit the properties of the next TextField, you could click on it in the Emulator. You can also choose a part from the Parts List, which is located in the middle of the button bar of the Composer. Select 'textField3' from the list. You will see the properties notebook for this part.
8. Change the **TextField label** from 'textField3' to 'Percentage'. Change the **Object name** to 'percentage'.
9. Choose the properties notebook of the button by selecting 'button4' in the Parts List.
10. Change the **Button text** from 'button4' to 'compute tip'.
11. Finally, choose the last TextBox, textBox5 from the Parts List.
12. Change the **TextBox text** to '\$tip' and the **Object name** to 'tip'.
13. Change the height of the TextBox to 15.

You have now set up the GUI for the Tip Calculator. If you decide that you need to change the location of some parts (for example, if the tip TextBox is too low on the screen) you can use the mouse to move the parts around so they all fit. After you have finished moving the parts around, go to the first part in the Parts List, TipCalculator, and uncheck the checkbox labeled 'Edit locations using mouse'. All the red squares in the Emulator will disappear, and the Emulator will now display your application exactly the way it will appear when run on the Palm OS Platform.

Responding to events

So far, you have created only the shell of your application. You can enter numbers into the TextFields, and you can press the button, but the button does not yet know what it is supposed to do.

On the Palm OS Platform, any part of the GUI can potentially respond to any pen up, pen down, pen move, or key down event which the Palm OS Platform receives. Simplicity lets you specify the Java source code to execute in response to these events.

Simplicity will dynamically execute the code that you write so that you can test your program as you design it.

There is only one part in the Tip Calculator which needs additional code written for it, and that is the 'compute tip' button.

1. Click the 'compute tip' button to show its properties in the Composer window.
2. Select the second page from the properties notebook, labelled 'penDown'. The code in this area will execute when the pen comes down on a button.

We will use the Code Sourcerer to write all of the code which is needed for the button. The Code Sourcerer will write Java code for you, based upon some simple choices.

3. Press the Code Sourcerer button toward the top of the 'penDown' page for the 'compute tip' button.
4. A dialog appears with a list of choices. We want to get the text in the 'amount' TextField. Choose the second item, labelled 'Ask a part about one of its properties...'. Press **Next**.
5. The Code Sourcerer now asks you which part you would like to query. Choose 'amount' (the TextField labeled 'Bill Amount \$' in your program). Press **Next**.

-
6. The Code Sourcerer now presents you with a list of the TextField's properties that can be queried. Choose 'TextField Text' if it is not already selected. Press **Done**.
 7. The Code Sourcerer now asks you where to store the value from the TextField. Replace the default choice with 'String amt'. Press **Ok**.

The following code should have appeared in the Pen Down Page.

```
String amt = amount.getText();
```

8. Next the Code Sourcerer will help you write code to ask the 'percentage' TextField for its contents. Press the Code Sourcerer button.
9. Choose the second option, 'Ask a part about one of its properties...'. Press **Next**.
10. Choose 'percentage' from the list of parts. Press **Next**.
11. Choose 'TextField Text' if it is not already selected. Press **Done**.
12. The Code Sourcerer now asks you where to store the value. Replace the default choice with 'String perc'. Press **Ok**.

The following new code should have appeared in the Pen Down Page.

```
String perc = percentage.getText();
```

Next you will need to add the code to do the calculation. The Code Sourcerer will generate this code for you as well.

13. Press the Code Sourcerer button.
14. Choose the option "Perform floating point computations...". Press **Next**.
15. You will now see a screen where you can enter a calculation which will be evaluated. You can enter it as straight text (for example, typing $2.5 * 2$), but in this case you want to see the result of a variable expression. Choose the third option, "The following variable expression".
16. You want to multiply the total amount of the bill by the percent tip. In the space to the right, type the following text:
`amt+"*"+perc+"/100"`
and press **Done**.
17. A last screen will appear asking where you wish to store the return value. The default value 'String result' is fine, so press **Ok**.

The following new code will appear on one line

```
String result = new com.datarp.calc.Calculator  
(String.valueOf(amt+"*"+perc+"/100")).evaluate();
```

Next, you will use the Code Sourcerer to make the button display this result in the tip TextBox.

18. Press the Code Sourcerer button.
19. Choose the first option if it is not already selected, 'Change a property of an existing part'. Press *Next*.
20. The Code Sourcerer now asks you which part you would like to change. Choose 'tip' (the TextBox which will display the amount of the tip). Press *Next*.
21. You will see a list of the properties which you can set in the TextBox. Choose the first option, 'TextBox text...'. Press *Next*.
22. The Code Sourcerer now asks where the text should come from. In this case, you want the text to come from the String which you already set up. Choose the third choice, 'The following variable expression'. Replace the text there with the following text:
"\$"+result
and press *Done*.

The following new code will appear

```
tip.setText(String.valueOf("$"+result));  
paint();
```

Some kjava parts will paint themselves whenever they change (like the TextField). The TextBox, on the other hand, will not do so. The Code Sourcerer has added the code to make the whole screen repaint.

Now you can test the code that you have just created. Type a number into the 'Bill Amount \$' field in your program. Type '15' into the 'Percentage field in your program. Then press the compute tip button. The amount of the tip is computed.

Creating an Exit button

Let's add one more thing to this program... a way to close it. By default, the user can exit by tapping the Home button. It's a good idea to have a button in your program that will exit as well. We will do this by creating a new button which will do this.

1. On the Object Palette, make sure that the first page, 'Kjava', is visible. (If not, click the mouse button on the 'Kjava' Tab.)
2. Click the first icon, named 'Button', with the mouse. (It should now appear depressed. Also, the title bar of the Object Palette will reflect your selection.)

-
3. Click on the right side of the Emulator, opposite the 'compute tip' button. You should see a new button appear on the screen.
 4. The new button's property sheet will be shown in the Simplicity Composer. Change the Button text from 'button6' to 'Exit'.
 5. Select the second page from the properties notebook, labelled 'penDown'.

We will use the Code Sourcerer again to write the code which will quit the application.

6. Press the Code Sourcerer button toward the top of the 'penDown' page.
7. A dialog appears with a list of choices. We want to quit the application, so choose the second option from the bottom, 'Java system operations...'. Press *Next*.
8. The Code Sourcerer asks you which system operation you want to do. You want to exit the program normally, so choose 'Exit the program with termination code' and leave the default value of '0'. Press *Done*.

The following code should have appeared in the Pen Down page.

```
System.exit(0);
```

This will cause the application to quit. When you press this button inside the emulator, you will see a window titled 'Program Terminated' with the message 'exit code 0' to let you know that it worked, but the Emulator itself will not actually be closed.

You have now finished designing your tip calculator. Save your work by choosing *Exit* from the **File** menu in the Composer. Choose *Yes* when asked if you want to save your changes.

Compiling the program

Looking back at the IDE, you should see the TipCalculator.Spotlet file in the 'Composer Files' group. Let's look at the code you just created.

1. Choose the 'Java Source Files' group. You should see a new file called TipCalculator.java. This is the java code which has been generated by the Composer. (If you are using the trial version of Simplicity for Palm OS Platform, the composer will not create the Java source file. Use Finished.java instead for the rest of the tutorial.)

2. Double-click the TipCalculator.java icon. This opens Simplicity's Java Source Editor with the TipCalculator.java file loaded. Browse through the code. You will see the code which you generated using the Code Sourcerer, in addition to code which Simplicity generated to set up the Spotlet.
3. Exit the Java Editor. (Don't save any changes to the file just yet.)

Now we want to compile and test the program inside the Emulator.

4. You want to bring up the pop-up menu for the TextEditor.java icon. This is usually done by clicking the right mouse button on the icon, but this behavior varies on different operating systems. You can also show the pop-up menu by holding the shift key and clicking the icon with the mouse.
5. Choose 'Compile' from the pop-up menu. Simplicity for Palm OS Platform now invokes your Java compiler to turn the Java source code which you just created into an executable file, called a Java class file. If you have not already run the J2ME Install Wizard, it will open now and help you to download the necessary files in the CLDC. The J2ME Install Wizard will also place the files in the proper place in your Simplicity install directory.
6. In the Project Tree, choose 'Java Class Files'. You should see an icon named TipCalculator.class. This file is your executable program.
7. Double click TipCalculator.class. The Class Viewer appears. The Class Viewer can provide a variety of information about the contents of a class file.
8. At the bottom of the Class Viewer is a large button labelled **Run**. Press this button to run the application.

Packaging your application

There are several steps which you must go through in order to put an application on a Palm OS Platform based handheld computer.

The first step is to build a .jar file with all of the files which are needed for your application. A jar file is useful because it will organize all of your files into one place.

1. Switch back to the IDE.
2. In the Project Tree, make sure that 'Java Class Files' is selected. Right-click on TipCalculator.class to bring up the pop-up menu. Choose 'Build jar...', located at the bottom of the list.

This will bring up the Jar Builder window. There are several ways that you can customize this window, but for this tutorial, you do not have to change any of the default settings.

3. Press the 'Build jar File' button. A Jar file of all the classes you need to run your application will be built for you.

Next you need to convert the Jar file into a Prc file. A Prc file is a file which can be loaded onto your Palm OS Platform based handheld computer.

4. Select 'Jar/Zip Files' in the IDE. Right click on TipCalculator.jar to bring up the pop-up menu. Choose 'Convert to PRC...'.
5. A new dialog will appear with many options for customizing the look of your application on the Palm OS Platform. For this test, the default settings are fine. Press the 'Convert' button at the bottom of the dialog.

Simplicity will now create a Prc file based on the Jar file you started with. Prc files are displayed in the 'Prc Files' group of the IDE. If you have set up your program settings correctly, you can double-click a Prc file from within Simplicity and it will be copied to your Palm OS Platform based handheld computer the next time you HotSync your device. You can run the program on the Palm OS Platform by tapping it with your stylus.

Congratulations! You have completed this tutorial and learned many of the basics of working with Simplicity for Palm OS Platform.

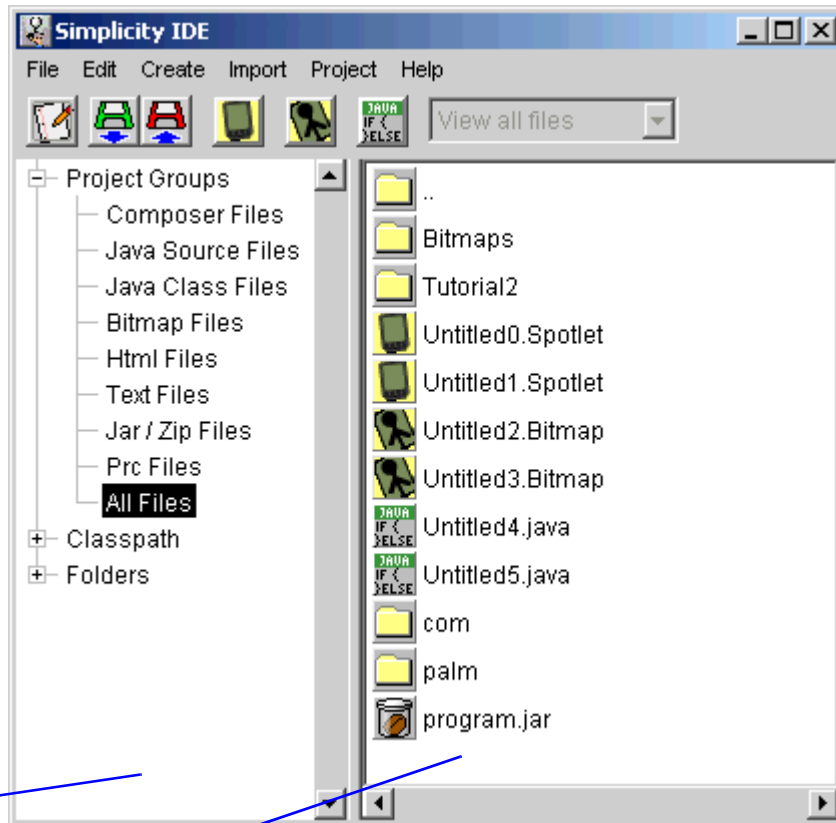
This chapter will discuss Simplicity's Integrated Design Environment (IDE) and how to use it to build and manipulate projects.

It will cover

- The IDE window, button bar, and menu bar
- File Groups
- Editing the Classpath
- Adding Folders
- Creating and importing files
- Program settings

The IDE Window

The first window which appears after Simplicity for Palm OS Platform is started is the IDE Window.



Project Tree

Group Contents Box

The IDE is centered around the Project Tree. The Project Tree organizes all of the elements in your project into three key areas:

- Project Groups
- Classpath
- Folders

The Classpath and the Folders areas will be discussed first, then the Project Groups.

The Classpath

The Classpath is where Simplicity looks for Java classes. It can include directories, Zip files, and Jar files. The Classpath should indicate directories at the root of the subtree of your java files. For example, a user wishes to access the class `com.datarp.calc.Calculator`. If the “com” directory is located inside a directory named `MyProjects`, then the entry on the Classpath would be similar to `/home/user/MyProjects`. Simplicity will search through the Classpath starting at the top, and stopping when the specified class is found.

Simplicity will first look for classes in the `CLDC` and `com.sun.kjava` packages. Simplicity will next check for classes on the dynamic Classpath.

Simplicity will generate package statements based on where a file is with respect to the Classpath. In the example above, files created in the “MyProjects” directory will not have any package statements. Files created in the “datarp” directory will be made part of the “datarp” package. Complex packages can be created by having folders inside of folders.

The Folders area

The Folders area is a place to put any directory that you want quick access to. It can be used to quickly traverse and view folders which are commonly used, or to find other files in the user’s file system.

Using the Classpath and the Folders area

It is possible to navigate the Classpath or the Folders area by expanding the nodes in the tree view of the directories which is displayed in the Groups List box area. A node with a minus sign is expanded. A node with a plus sign is collapsed.

Items may be added or removed from the Classpath and the Folders area using the Classpath editor and the Folders editor, both found on the Edit Menu in the IDE.

Project Groups

A project can also be organized into groups of related items. The Groups in a project are listed in the first area of the Project Tree and can be modified by choosing *Edit Groups* from the *Edit* menu

A group consists of fields.

1. Group Name
2. Group Directory
3. Group Extensions

The user can edit any Group and modify the directory to be searched and the filename extensions to be included. The directory may be specified relative to the project's directory, or as a fully qualified path. This allows a project to access files located anywhere on the user's file system. Any number of extensions may be specified. If no extensions are specified, all files in the directory are included as well as the parent directory ('..') allowing complete file system navigation.

Groups can also be used to filter the files which are viewed in the Classpath or the Folders area. If a group is chosen from the drop-down menu on the menubar, only files with the appropriate extensions will be viewed. This can be useful when directories with many files in them are being viewed.

Editing parts of the Project Tree

The Project Groups, the Classpath, and the Folders area can all be edited in the IDE.

Editing Groups using the IDE Group Editor

The IDE Group Editor lists the current groups. When the user chooses a Group, the group's fields are displayed. The values can be edited in the entry fields on the right of the Group Editor.

The following commands are available.

- **New** Create a new Group, initially named 'new group'.
- **Up** Move the selected Group one position upward in the Project Tree.
- **Down** Move the selected Group one position downward in the Project Tree.
- **Delete** Erase the selected Group.
- **Ok** Accept the changes.
- **Cancel** Dismiss the changes.

Editing The Classpath

The Classpath Editor lets the user manipulate the items on the Classpath and add new directories or Jar/Zip files. The following commands are available.

- **Up** Move the selected directory or Jar/Zip file one position upward in the Classpath.
- **Down** Move the selected directory or Jar/Zip file one position downward in the Classpath.
- **Remove** Remove the selected directory or Jar/Zip file from the Classpath.
- **Add directory** Opens a new dialog to choose a directory to add to the Classpath. The current directory is shown at the top of the dialog. The special files “..” and “.” are displayed. It is possible to move to the parent directory by double-clicking on the “..” file. The “.” file signifies the current directory.
- **Add Jar/Zip file** Opens a system-dependent dialog to choose a Jar or Zip file to add to the Classpath.
- **Ok** Accept the changes to the Classpath.
- **Cancel** Dismiss the changes to the Classpath.

Editing The Folders area

The Folder List Editor lets the user manipulate the items in the Folders area and add new directories or Jar/Zip files. The Folder List Editor is similar to the Classpath Editor. The following commands are available.

- **Up** Move the selected directory or Jar/Zip file one position upward in the Folders area.
- **Down** Move the selected directory or Jar/Zip file one position downward in the Folders area.
- **Remove** Remove the selected directory or Jar/Zip file from the Folders area.
- **Add directory** Opens a new dialog to choose a directory to add to the Folders area. The current directory is shown at the top of the dialog. The special files “..” and “.” are displayed. It is possible to move to the parent directory by double-clicking on the “..” file. The “.” file signifies the current directory.
- **Add Jar/Zip file** Opens a system-dependent dialog to choose a Jar or Zip file to add to the Folders area.
- **Ok** Accept the changes.
- **Cancel** Dismiss the changes.

Opening items in the Group Contents Box

Choosing any item in the Project Tree displays its contents in the Group Contents Box.

The Group Contents Box shows the name of each item in the group alongside an icon which indicates the type of item and the behavior of the item when opened. The Group Contents Box is aware of and has special behaviors for

- Composer Files (*.Spotlet)
- Bitmap files (*.Bitmap)
- Java source files (*.java)
- Java class files (*.class)
- Java archives (*.zip, *.jar)
- Text and HTML files (*.txt, *.html, *.htm)
- Java Beans
- Folders

Each item has a pop-up menu associated with it which contains those actions that are appropriate for the selected item. For example, a Java class file's pop-up menu has the default options: Open, Copy, Rename, and Delete as well as an option to invoke the compiler.

The first action on an item's popup menu is always Open. This action can also be performed by double clicking the item.

- Opening a Composer file will launch the Simplicity Composer. This file stores the contents of a Spotlet.
- Opening a Bitmap file will launch the Bitmap Editor. A Bitmap file stores the data of a bitmap in a text format which may also be edited by a text editor outside of Simplicity.
- Opening a Java source file will launch the Simplicity Java Editor. This default action may be overridden in the Program Settings in order to specify a preferred Editor.
- Opening a Java class file will launch the Simplicity Class Viewer which will allow inspection of the contents of that class. The Class Viewer will also include a button which will execute the Spotlet's `public static void main(String[])` entry point.

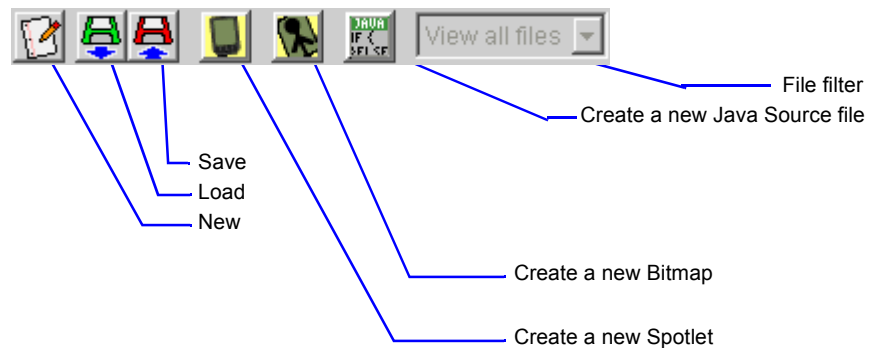
- Opening an HTML file will launch the preferred web browser specified in the Program Settings.
- Opening a folder will display the contents of that folder in the Group Contents Box. Folders only appear when all files are being displayed.
- Opening all other files will launch a Text Editor.

IDE Menu Bar

The IDE Menu Bar provides a set of commands for manipulating items in a project. The Menu Bar also provides commands for opening, saving, and closing projects as well as exiting from Simplicity and getting help.

IDE Button Bar

The IDE Button Bar provides easy access to the most commonly used menu items.



File Menu

The File menu has the following options.

- **New Project...** This option will prompt the user for the name of a new project to create, and the project's main directory. The main directory can be anywhere the user chooses. A **Browse...** button is available to help the user choose the location. If needed, the project's main directory will be created at the specified place. Once the main directory is specified, Simplicity will then create a set of

directories for the new project. If a project was previously open, the user will be given the option of saving any changes before it is closed.

- **Open Project...** This option will prompt the user to select one of the projects that has previously been created. Projects may be anywhere the user wishes.
- **Save Project...** This option will save any changes to the current project.
- **Close Project...** This option will close the current project. The user will be given the option of saving any changes before it is closed.
- **J2ME Install Wizard...** This option will open up a window which will step the user through installing J2ME so that Simplicity is able to properly compile classes. The Wizard will open automatically when a user attempts to compile if it has not already run.
- **HotSync KVM...** This option will HotSync the KVM and KVMUtil to the user's Palm OS Platform based handheld computer. Note that the HotSync utility must have been set up in the Program Settings.
- **Migrate Old Projects...** This option will allow Simplicity to import projects from earlier versions of Simplicity. The user may choose a Project directory, and then pick which projects to import. After using this option, the projects will appear in the list given by **Open Project...** To share projects from Simplicity 1.2 or later, simply copy the .Simplicity file corresponding to the desired project into the Personal Settings Directory.
- **Program Settings...** This option will launch the Program Settings Dialog (page 31) which allows the user to modify many aspects of how Simplicity for Palm OS Platform operates. All changes are applied after the user presses the **Ok** button and affect all projects.
- **License...** This option will bring up a license information window. The window will display the type of license which is currently active and, if appropriate, the name and company of the user. The default is a "Tryout" license. The window will display a text area where the user may enter a new license key if the "Enter License Information" button is pressed.
- **Show Console/Hide Console** These options will make visible or hide the Java Input/Output Console.
- **Exit** This option will exit from Simplicity for Palm OS Platform. If a project was previously open, the user will be given the option of saving any changes before it is closed.

Edit Menu

The Edit menu has the following options. Those options which modify items will make permanent changes to files on your hard drive.

- ***Open Selection*** This option will open the item in the Group Contents Box which has been selected with the mouse. This has the same effect as double clicking an item or choosing ***Open*** from an item's pop-up menu.
- ***Close Selection*** This option will close the item in the Group Contents Box which has been selected with the mouse.
- ***Rename Selection*** This option will allow the user to change the name of an item. A text field will appear in place of the item containing the old name. Pressing the mouse button anywhere outside the text field will accept the change. This has the same effect as choosing ***Rename*** from an item's pop-up menu.
- ***Copy Selection*** This option will make a copy of an item. This has the same effect as choosing ***Copy*** from an item's pop-up menu.
- ***Delete Selection*** This option will delete an item. This has the same effect as choosing ***Delete*** from an item's pop-up menu.
- ***Refresh*** This option will cause the Group Contents Box to update itself with any changes that have been made to the project directory. This may be necessary when external programs create or modify files in a project.
- ***Edit Groups...*** This option will launch the Group Editor. This will allow the list of groups to be modified.
- ***Edit Classpath...*** This option will let the user interactively add folders and/or Zip files to the Classpath. They will be available ***immediately***. They will also be shown immediately in the IDE's main window.
- ***Edit Folders...*** This option will let the user interactively add folders to the IDE's main window. This can be a useful way to access files which do not have to be in the Classpath.

Create Menu

The Create menu has the following options. The user may need to select an appropriate group in order to see the new item.

- ***Spotlet*** This option will create a new Spotlet Composer file. This will allow the user to create a new java program for the Palm OS Platform.

- ***Bitmap...*** This option will create a new Bitmap after the user is prompted for the desired height and width. Note that Bitmaps must be a multiple of 16 pixels wide. Once a Bitmap has been created, it can be edited with the Bitmap Editor (see page 42)
- ***Java File*** This option will create a new empty Java source file.
- ***Text File*** This option will create a new empty text file.
- ***Data File*** This option will create a new empty binary data file.
- ***HTML File*** This option will create a new empty HTML file.
- ***Folder*** This option will create a new folder.

Import Menu

The Import menu has the following options. The user may need to select an appropriate group in order to see the new item.

- ***Text File*** This option will import a text file.
- ***Data File*** This option will import a binary data file.
- ***HTML File*** This option will import an HTML file.

Project Menu

The Project menu has options for operations affecting an entire project

- ***Compile project*** Compile all Java source files in the Project directory.
- ***Compile current group*** Compile all Java source files in the currently selected group.
- ***New command window...*** Opens a new Java Command window (see page 73).

Help Menu

The Help menu has the following options.

- ***User Guide...*** This option will show the user a hyper-linked table of contents of the Simplicity for Palm OS Platform User Guide (This book!).
- ***Tutorials...*** This option will show the user the table of contents for the Simplicity for Palm OS Platform Tutorials.
- ***About Simplicity...*** This option will show the user the Simplicity for Palm OS Platform 'About Box' and version information. Clicking the mouse button anywhere will dismiss the about box.

Program Settings

The program settings dialog has five pages.

- Directories
- External Editors
- Object Palette
- Java Editor
- Printing

Directories

The Directories page contains the following fields.

- ***Simplicity Installation Directory*** This is the directory where Simplicity's core files are stored. You cannot change this information here.
Example: C:\Program Files\Simplicity
- ***Personal Settings Directory*** This is the directory where the information that Simplicity creates about projects will be stored. The personal settings file and other files that Simplicity needs will also be stored here. You cannot change this information here.
Example: C:\Program Files\Simplicity\simplicitypalms
See page 6 for more details.
- ***Preferred Java Compiler*** If this field is left blank, then Simplicity for Palm OS Platform will attempt to use the Java compiler which is built into your platform's Java Development kit. The user can override this behavior to use a different compiler by specifying a command here. For most purposes it is preferable to leave this field blank.
Example: C:\jdk1.3\bin\javac.exe
- ***Preferred JDB Command*** If this field is left blank, then Simplicity for Palm OS Platform will attempt to use the JDB debugger which is built into your platform's Java Development kit. The user can override this behavior to use a different JDB-compatible debugger by specifying a command here. In most cases this field should be left blank, as the debugger is very sensitive to different environments.
- ***Show Opening Dialog*** If checked, a welcome screen will appear when Simplicity is first started, presenting the user with three choices: Open an existing project, create a new project, or view the tutorials.

External Editors

The External Editors page contains the following text fields.

- ***Pre-verifier Executable*** This field must have the location of a Java preverifier executable. Any valid preverifier may be entered here. If the J2ME Install Wizard has successfully run on Windows or Solaris, this field may be left blank.
Example: C:\jdk\bin\preverify.exe
- ***PRC Install Tool Executable*** This field may have the location of a tool which will install a PRC onto a Palm OS Platform based handheld computer.
Example: C:\Program Files\Palm Desktop\instapp.exe
- ***Preferred Web Browser*** This is the full path to the web browser in which the user has chosen to view HTML pages.
Example: /usr/local/bin/netscape
- ***Preferred Java Editor*** If this field is left blank, then Java source files will be edited with Simplicity for Palm OS Platform's Java Editor. Otherwise, the user may specify their favorite editor here.
Example: /usr/bin/vi
- ***Preferred Text Editor*** If this field is left blank, then text files will be edited with Simplicity for Palm OS Platform's Text Editor. Otherwise, the user may specify their favorite editor here.
Example: C:\Win95\notepad.exe

Object Palette

Video display size and quality can vary greatly between computers. The Object Palette Customizer lets the user adjust the size and behavior of the Object Palette so that it will be clear and easy to use without wasting unnecessary space on the user's video display. A small sample palette is provided, so that the user may observe the effect of changes immediately. The following settings may be adjusted.

- ***Spacing*** The icons in the palette will be spaced this many pixels apart.
- ***Shadow*** The icons will cast a shadow this many pixels deep. The depressed icon will cast a shadow half this number.
- ***Enable Flyouts*** If selected, icons will popup a small window containing the text of the icon. This is particularly useful if Pictures Only is selected.
- ***Show palette as*** These three radio buttons allow the user to indicate that icons should be shown using pictures, text, or both.
- ***Size*** This slider lets the user indicate how large icons should be. This is very useful for small displays.

- **Scaling Algorithm** This choice field lets the user choose which algorithm to use to scale the icons. While “Smooth Scaling” is often the best choice, the choice which appears most attractive may vary on different platforms.

Java Editor

The Java Editor Customizer lets the user customize how Simplicity for Palm OS Platform’s Java Editor should operate. A small sample editor is provided so that the user may observe the effect of changes immediately. The following settings may be adjusted.

- **Background** The background color.
- **Foreground** The color for class names and variable names.
- **Keywords** The color for Java Keywords (e.g. `int`, `for`, `class`, `void`,...).
- **Numbers** The color for numbers (e.g. `3`, `3.14`, `3e-5`, `0x0011`,...).
- **Strings** The color for text strings (e.g. `"hello"`, `"\""`,...).
- **Characters** The color for characters (e.g. `'a'`, `'\\'`, `'\011'`,...).
- **Operators** The color for Java operators (e.g. `+`, `-`, `=`, `==`, `[]`,...).
- **Comments** The color for comments (e.g. `/* comment */`).
- **Errors** The color for lexical errors (e.g. `bad@name`).
- **Enable Auto-tab** Enabling this option inserts tab characters when the Return/Enter key is pressed so that the new line lines up with the previous line.
- **Substitute spaces for tabs** Enabling this option will replace tabs with spaces, the number of which is determined by the **Tab Size**.
- **Font** The display font.
- **Tab Size** The number of characters in each tabstop. The width of an ‘n’ character of the chosen font is used to compute the tab stop width.

Printing

This settings page complements the **Java Editor** page by providing settings specific to printing Java Source code to a printer. A preview is provided to serve as a guide to the appearance of the printed page.

- **Margins** These four fields specify the top, bottom, left, and right margins on the printed page.
- **Tab size** This field specifies the size, in inches, of tabs. This can have a large effect on the printing of source code.

- ***Wrap long lines*** If selected, lines which extend beyond the right margin will be wrapped to the next line. This option may make printed output appear less attractive, though it ensures that all text is printed.
- ***Print page headers*** Select this option to print a header on the top of each page. The header will include the filename and the page number. The first page will also include the current date.
- ***Printer font*** The font to be used for printing

This chapter will discuss Simplicity's Java Source Code Editor. The Editor is designed to provide many powerful tools for working writing Java code, while being easy to learn and use.

This chapter will cover:

- Basic Editing
- Printing
- Search and Replace
- Using the Sourcerer's Apprentice
- Configuration

Editing

The general design of the Java Source Editor is intended to be familiar for users. General text editing, cursor movement with the keyboard and mouse, as well as selecting text using the mouse all operate in the standard fashion as most other text editors.

The Java Source Editor can be used as a standalone editor within the Simplicity IDE. It is also used within the Composer anywhere that code areas appear. The standalone editor has a menubar at the top for accessing its features, as well as a pop-up menu containing most features. When the editor is used in a Composer, only the popup-menu is available. In addition most menu items have keyboard shortcuts available.

File Menu

- **New** clears any text in the editor in order to start a new document. The user will be prompted to save any unsaved changes.
- **Open...** displays a dialog for the user to select a new file to edit. The user will be prompted to save any unsaved changes.
- **Save** saves the current document.
- **Save As...** saves the current document to a new file name.
- **Print...** lets the user print the current document. If the printer supports color, the color-syntax display will be printed in color. This item first displays an operating system-specific print dialog, allowing the user to customize the print job.
- **Exit** closes the editor. The user will be prompted to save any unsaved changes.

Edit Menu

- **Undo** undoes the last operation. This can be used as many times as the user wishes to undo changes. All operations in the editor can be undone.
- **Redo** is the opposite of Undo. Once an edit has been undone, it can be redone until another edit is performed.
- **Cut** removes the selected text from the editor and places it on the system clipboard.
- **Copy** copies the selected text to the system clipboard.
- **Paste** pastes any text in the system clipboard into the editor at the cursor.
- **Delete** deletes the selected text.
- **Select All** selects all text in the editor.
- **Sourcerer's Apprentice...** displays the methods and fields in a particular class. See Sourcerer's Apprentice on page 39.
- **Goto...** displays a dialog letting the user jump to a specific line in the code.

- ***Search & Replace...*** displays a dialog letting the user search for some text and optionally replace it with new text. See Search & Replace.

To facilitate fast editing, the following keyboard shortcuts are available:

TABLE 1.

Feature	Keyboard Sequence	Alternate Sequence
Cut	Ctrl-X	Shift-Delete
Copy	Ctrl-C	Ctrl-Insert
Paste	Ctrl-V	Shift-Insert
Delete	Delete	
Select All	Ctrl-A	
Undo	Ctrl-U	Ctrl-LeftArrow
Redo	Ctrl-R	Ctrl-RightArrow
Goto...	Ctrl-G	Ctrl-J
Search & Replace...	Ctrl-F	Ctrl-S
Sourcerer's Apprentice	Ctrl-Space	F1

Holding down the Shift key while using the arrow keys can also be used to select text using the keyboard.

MacOS users should note that the Command (cloverleaf) key may be used instead of the control key for all of the above shortcuts.

Indentation Features

The Java Source Editor can optionally auto-indent when the user presses the Enter key. This saves time by automatically adding the same number of tabs at the beginning of a new line as the previous line.

The user can also change the indentation level of a block of code by selecting the code and then pressing tab. The entire block will be indented. Shift-Tab will remove an indentation tab from the block.

The tab size and the auto-indent feature can be configured in the IDE's program settings.

Color and Printing Features

The Java Source Editor can identify various Java language elements by displaying each in a different color. The language elements that it can identify are: keywords, number, strings, characters, operators and comments. Each of these may be assigned an individual color in the Program Settings. Additionally, the background and foreground colors can be specified, and a screen font can be chosen.

When printing a Java source file, all of the color settings (with the exception of background) will be used if the printer supports color printing. The user can also specify page margins and a printer font in the Program Settings' Printing page. The user can also indicate whether long lines should be wrapped (they will be wrapped at word boundaries), and whether a header should be printed at the top of each page.

To print a Java source file, select the print command on the File menu of the Java Source Editor. An operating system-specific dialog will appear displaying any options for the available printer(s). After the user confirms this dialog, the document will be printed.

Search & Replace

The Java Source Editor's 'Search & Replace' dialog lets the user find any occurrences of a text string and optionally replace them with a new text string. A single 'Search & Replace' dialog is shared between all Java Source Editors, making it easy to perform the same search on multiple files. The dialog will always perform the search in the Editor which currently has the focus.

The search can be performed in three modes:

- **Ignore case** finds any matches with the same letters, but ignoring the case of the letters. For example, 'java' would find occurrences of 'java', 'JAVA', and 'jAvA'.
- **Match case** finds any matches with the same letters. For example, 'java' would find only occurrences of 'java', and not 'JAVA' or 'jAvA'.
- **Match Perl 5 regular expressions** finds matches based on the regular expressions syntax of the Perl 5 language. For example, '[a-c]*' matches any word starting with 'a', 'b', or 'c'.

The following buttons are available for searching:

- ***Find Next*** finds the next occurrence of the string in the ‘Find what:’ text field. The search is always performed starting at the cursor location. If a match is found, it will be highlighted in the editor, otherwise a beep sound will indicate no match.
- ***Replace*** will replace the highlighted text in the editor with the text in the ‘Replace with:’ text field. It will then automatically perform a find operation.
- ***Replace All*** will replace all occurrences of the search string with the replacement string.
- ***Cancel*** will dismiss the Search dialog.

All replacement operations can be undone in the editor.

The Sourcerer’s Apprentice

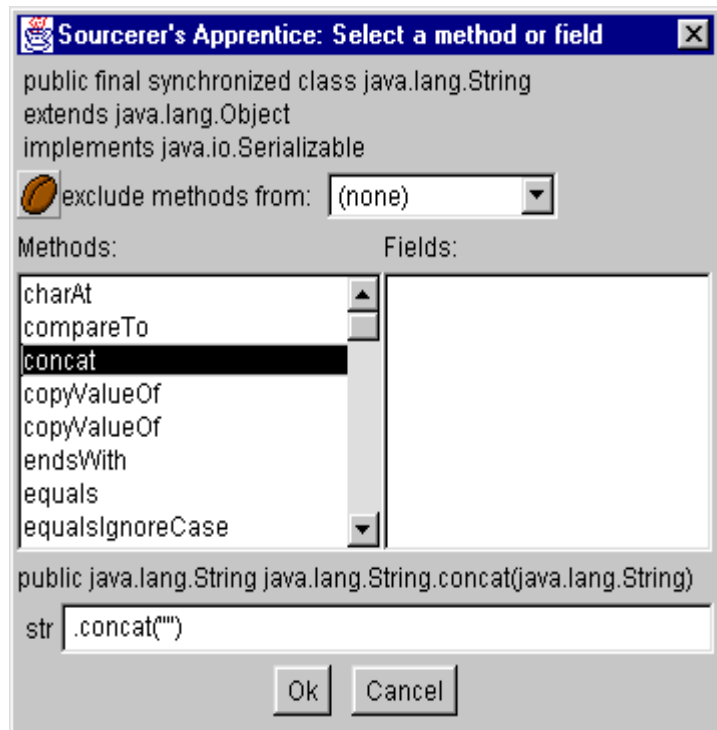
The Sourcerer’s Apprentice lets the user quickly view and select from the available methods and fields in a class. This feature is invoked from the menubar or popup-menu, or by pressing Ctrl-Space or F1. When this is done, the Sourcerer’s Apprentice takes the Java identifier to the left of the cursor and determines its class. It then displays a dialog showing all the methods and fields in that class. Note that the methods and fields shown will be for the CLDC or kjava classes, which may have fewer methods and fields than their J2SE equivalents. The user can select a method or field which will then be displayed at the bottom of the dialog. If a method is selected, default arguments will be provided. The user can edit this code if they wish. When the OK button is pressed, this code will be inserted in the editor at the cursor’s position. The Cancel button dismisses the dialog without making any changes to the code.

For example, if a section of code has a String object named ‘str’, typing str and then pressing Ctrl-Space will launch the Sourcerer’s Apprentice. See Figure 2.

Figure 2. Example of Sourcerer's Apprentice

```
String str;  
str
```

← *Typing Ctrl-Space here shows all
of the methods and fields in 'str'*



If the Sourcerer's Apprentice is invoked next to something which is not an instance of class, it will beep, but not display a dialog box.

This chapter will discuss the Bitmap Editor.

This chapter will cover

- Using bitmaps within a spotlet
- Editing bitmaps with the Bitmap Editor

Using bitmaps within a Spotlet

A Bitmap is a simple black and white picture which can be displayed in the Palm OS Platform. Bitmaps can be any height, but must be a multiple of 16 pixels wide.

A Spotlet can draw bitmaps in several ways. One way is to use the kjava Graphics object to draw a bitmap directly on the screen of the Palm OS Platform Emulator. The following code fragment will draw a Bitmap, assuming that the integers “left” and “top” and the Bitmap “bitmap” have already been set up. (Simplicity defines g as a reference to the kjava Graphics object in every Spotlet.)

```
g.drawBitmap(left, top, bitmap);
```

However, most users will want to use a Bitmap in a different way: as the picture for a button. When a button is created, it can be given a Bitmap which it will draw instead of a label. A third way of using a Bitmap is as an icon for the application on the Palm OS Platform.

Using the Bitmap Editor

However the Bitmap will ultimately be used, the first step is to create a new Bitmap file in the IDE. This can be done using the Button Bar, or from the “Create” menu in the IDE. The user will be prompted for the size of the new Bitmap. This cannot be changed by the Bitmap editor once it is set.

After a new Bitmap file is created, it may be opened in the Bitmap Editor. The Bitmap Editor is a quick way to edit bitmaps. The grid in the center of the Bitmap Editor is an expanded view of the bitmap. The actual size of the Bitmap is displayed at the top of the window.

Tools

There are six tools in the Bitmap Editor. They are

- ***pencil*** This tool draws one pixel at a time in the selected color on the grid.
- ***line*** This tool draws a line in the selected color on the grid. Before the line is drawn, the squares which will be effected show up in an alternate color to hilight what will be drawn. Press the mouse to select the starting position, and drag to the end position.
- ***oval*** This tool draws an oval in the selected color on the grid. Press the mouse at the center of the oval, and drag to the desired size. If the control key is held down, the tool will draw a circle.
- ***filled oval*** This tool is identical to the oval tool, except the shape drawn will be filled with the default color.
- ***rectangle*** This tool draws a rectangle in the selected color on the grid. Press the mouse at one corner and drag to the desired size. If the control key is held down, the tool will draw a square.
- ***filled rectangle*** This tool is identical to the rectangle tool, except the shape drawn will be filled with the default color.

The default color is chosen in two ways: by the mouse button pressed when clicking, and by the color associated with each mouse button. Mac OS users note: the alt/option key is used to simulate mouse button 2, not the control key. The default color can be switched from black to white and vice versa by clicking on the appropriate field on the right side of the window.

To save a Bitmap, close the Bitmap Editor window using the default method for your OS. You will be asked if you wish to save the file.

A Bitmap file is saved in a text-only format which can be opened by a text editor. They are saved in the form of a Java code snippet which could be pasted into some existing Java source code. Simplicity will automatically import the correct code from a Bitmap file for you when used with a button in the Composer.

The Spotlet Composer

This chapter will discuss the Spotlet Composer. The Composer lets the user specify properties of any parts, as well as enter user code. It will also discuss how to create a program which can run on the Palm OS Platform.

This chapter will cover

- Choosing and creating composers
- The Composer window, button bar, and menu bar
- Property notebooks
- Spotlet properties and methods
- Code and application generation

Creating a New Composer

A new Spotlet Composer can be created by selecting the **Create Spotlet** menu item from the **Create** menu in the IDE. A Spotlet Composer can also be created by pressing the Create Spotlet button on the tool bar.

The Spotlet Composer

A Spotlet Composer is used to create a Java Application which can run on the Palm OS Platform. Spotlets are similar to Applications which are used in “Standard Edition” Java, but have special methods which are used to provide functionality for the Palm OS Platform.

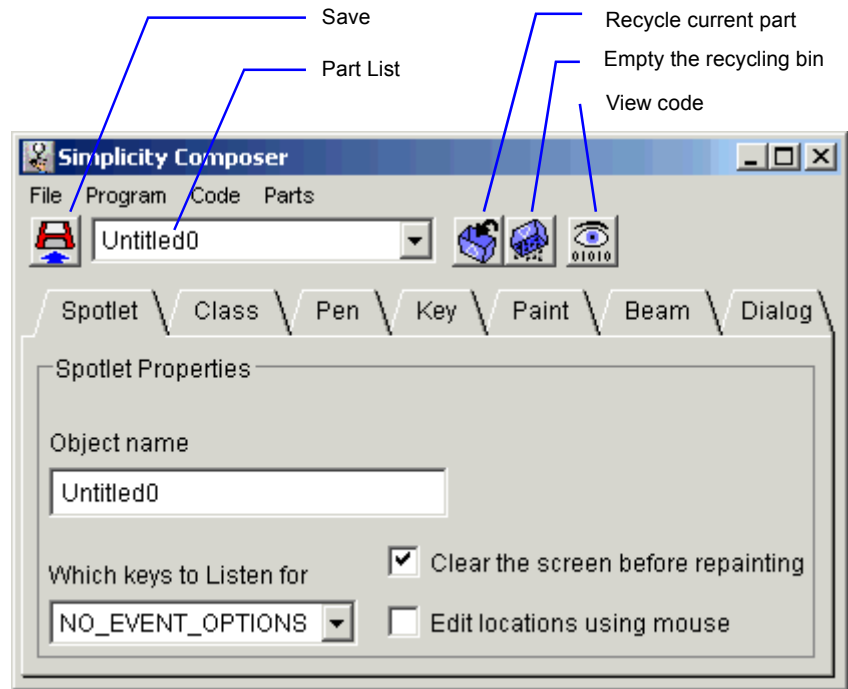
A Spotlet composer will generate a single Java source file corresponding to a public class.

When a Composer is opened from within the IDE, three windows appear. These three windows are the Composer window, the Object Palette, and the Palm OS Platform Emulator window.

The Composer Window

The Composer window is a dynamic, context-sensitive property editor in which the user can assign default properties and behaviors for all parts in a program. All user supplied code that a program requires is entered into the composer, including event response code, class declaration and constructor code, and method code which can be called from within the class or from external code which will use the class.

The Composer window contains a menu bar, a button bar, and a context sensitive set of tabbed pages.



Composer Button Bar

The Composer button bar contains the following items:

- **Save** This button is identical to the Save option on the file menu. It saves the current state of the composer to disk.
- **Part List** This choice field contains a list of all of the parts being used in the current program. The user can view the properties of any part by selecting it from the Part List.
- **Recycle Current Part** Pressing this button removes the part currently displayed in the Part List from the emulator and stores it in the Recycled page in the Object Palette. The part is still active while in the Recycled page. It can be modified from its properties page as well as referenced by event code.
- **Empty The Recycling Bin** Pressing this button deletes all parts from the Recycled parts page. The parts are no longer accessible from event code.

- **View Code** Pressing this button opens the Code Viewer window, which shows the current generated code. If the window is already visible, pressing this button will update the code and bring the window to the front.

File Menu

The File menu has the following options.

- **Save** This item saves the current state of the composer to disk, as well as the generated Java code.
- **Generate code** This item saves the current generated Java code, but does not save the current state of the composer.
- **View code** This item is equivalent to pressing **View Code** in the button bar.
- **Exit** This item exits the Composer. It will query the user to save any changes first. Upon exiting, a Java source file will be created containing all of the generated code.

Program Menu

The Program menu has the following options.

- **Command window...** This item opens a new Java Command Window (see page 76).
- **Initialize Class** This item restores all parts to the current settings in the Composer window. The Emulator is restarted, and any code in the Code Declaration and Constructor pages is also executed.

Code Menu

The Code menu has the following options.

- **Goto declaration code** This item will bring the declaration code area into view in the Composer window. The user should use this code area to declare any instance variables as well as static variables.
- **Goto constructor code** This item will bring the constructor code area into view in the Composer window. The user should use this code area for any code that should be added to the class constructor.
- **Goto method code** This item will bring the method code area into view in the Composer window. The user should use this code area for any instance methods or static methods that should be included in the class.

Parts Menu

The Parts menu has the following options.

- **Recycle current part** This places the part currently displayed in the Part List from the Emulator and stores it in the Recycled page in the Object Palette, just as if the button with the same name had been pressed.
- **Empty Recycling bin** This deletes all parts from the Recycled parts page, just as if the button with the same name had been pressed.
- **Enable Events** When this item is unchecked, parts in the Emulator will not execute user code in response to events. This will allow you to temporarily suspend most code execution, to allow easier editing. Be sure to re-enable this checkbox to continue testing your code.

Property Notebooks

Notebook pages

Each time a part is added to the Emulator, its Properties Notebook is displayed in the Composer window. At any future time, a part's Properties Notebook can be viewed either by selecting the part's Object Name from the Part List or by clicking the part with the mouse.

Every part in the Composer has a minimum of one page in its properties notebook. This page is the part-specific properties page.

All properties specified in the properties notebook are the initial properties that a part will have. Some properties may be changed in the Emulator by either event code or by user interaction with the Emulator. The user may restore all parts to the state specified in the properties pages by choosing the **Initialize Class** menu item on the **Program** menu.

The property page for any part contains the name which will identify the declared object and the part-specific properties for that object. For example, the first page for a Checkbox would contain the Object Name plus the checkbox text and the initial checkbox state.

Parts may also have additional pages, where users can type code which executes when a part receives a penDown, penMove, or keyDown event, in addition to the default behavior for the part.

For example, when a button receives a penDown event, it will briefly highlight itself. In addition to this default behavior, users can write code which will occur after the highlight.

The user can enter any legal Java statements in these code event areas. Simplicity watches for any changes made and updates the Emulator so that the Emulator always reflects the current state of the code. If there is an error in the code, a message will be printed to the console. If an import statement is entered in a code event area, it will be moved to the proper place in the generated code.

At the top of every code page is a ***Code Sourcerer*** button. The Code Sourcerer will write Java statements for the user based upon the user's choices.

Spotlet Properties

A Spotlet has several special additions to its property page. They are the

- Event Options choicebox
- Clear screen checkbox
- Mouse editing checkbox

Event Options

This choicebox allows the user to specify which events the Spotlet will respond to. The choices are

- ***WANT_SYSTEM_KEYS***
- ***NO_EVENT_OPTIONS***

WANT_SYSTEM_KEYS, the default, will capture all events, including

- Page Up/Page Down keys
- Hard Keys (Calendar, Phone, etc.)
- Soft Keys (Home, Calculator, etc.)
- Power Key
- Graffiti(TM) input

- HotSync(TM) button

Spotlets which implement this choice should have a programatic way to exit the Spotlet.

NO_EVENT_OPTIONS will capture fewer events, letting the Palm OS Platform handle the remaining. The events captured are

- Page Up/Page Down keys
- Graffiti(TM) input

Clear Screen

This checkbox will clear the screen before repainting.

Edit locations using mouse

This checkbox enables the editing of the position and size of parts in the Emulator using the mouse. A red square will be drawn around each part in the Emulator to facilitate editing.

Spotlet code areas

The properties notebooks for the Spotlet also have a page titled ***Class***. This page contains code areas for user code. These code areas include

- ***declaration code*** This code area should be used to declare any static variables or instance variables that a program may need. Any Java statement which is legal to exist within the outermost scope of a Java class can be put here.
- ***constructor code*** This code area should be used for any additional initialization code that a class may need. Any legal Java statements may be placed here. This code will be appended to the class constructor.
- ***method code*** This code area should be used to declare any static methods or instance methods that a program may need.

A Spotlet Composer will also have code areas for the special methods of a Spotlet. These code areas include

- ***pen events*** This code area can be used to write code which will execute whenever a ***penDown***, ***penUp***, or ***penMoved*** event occurs.

- **key down** This code area can be used to write code which will execute whenever a key is pressed. Key presses include text entry via Graffiti(TM) as well as the hard buttons (e.g. calendar, phone...) and the soft buttons (home, calculator...) on a Palm OS Platform based handheld computer. New Spotlets will have several lines of code in this area. The code may be used to perform different tasks depending upon which key has been pressed. This code has been set up to exit the program when the “Home” key is pressed. This code may be changed or deleted as desired.
- **paint** This code area can be used to add code to the paint convenience method which will be added to every Spotlet that Simplicity creates.
- **beam receive** This code area can be used to write code which will execute whenever the Spotlet receives a “beam” event, or infrared signal from another Palm OS Platform based handheld computer. This event can be simulated by choosing ‘Send Beam Event’ from the *Options* menu of the Emulator.
- **dialog dismissed** This code area can be used to write code which will execute whenever the Spotlet regains control from a Dialog.

Many of these code areas can be also be reached using the *Code* menu in the Composer.

Code and Application Generation

After Simplicity has generated source code, there are several steps which are needed to get that code running on a Palm OS Platform based handheld computer. They are

1. Compiling
2. Building a jar
3. Converting to PRC
4. Loading onto a Palm OS Platform based handheld computer

Compile against the J2ME

The source code which Simplicity generates is created according to the J2ME specification. You must compile the .java files which are created by the Spotlet Composer against the J2ME for the compiling to work correctly. To do this within Simplicity, right-click (or control-click) on a source file in the IDE and choose ‘Compile’ from the popup menu. If the J2ME Install Wizard has not already been

run, it will open up and help you to download and install the necessary components. If you wish to compile outside of Simplicity, you will need to issue a command from the command line of your OS which looks something like

```
javac -g:none -d tmp -classpath %allclasspath% -bootclasspath  
%j2meclasspath% *.java
```

Where %allclasspath% is the classpath which includes your Spotlet, as well as your J2SE classes, and %j2meclasspath% is a classpath which only includes the J2ME CLDC classes.

Generate a Jar file

To invoke the Jar Builder, select a .class file in the IDE which contains the first Spotlet which will be run by your application. Choose ‘Build jar...’ from the popup menu which can be obtained by right-clicking or control-clicking the file.

In the ***Jar Builder*** tool the following options are available

- set starting points
- specify packages to exclude
- view dependencies
- compress files
- change jar file name

Class names entered in the ***Starting Points*** area will be examined to determine all dependencies which will be needed for the jar file. If there are packages which are known to be available on the deployment machine, they can be excluded in the ***Exclude packages*** area. Enter the names of packages to be excluded and a trailing period; for example “java.” or “com.sun.”. The ***View Dependencies*** button will create a tree which shows which classes rely on other classes (note that classes loaded by the forName() method will not appear here and must be added manually.)

Convert to PRC

Before a jar file may be run on the Palm OS Platform, it must first be run through a “PreVerifier” and then converted into a “PRC” file. To do this, right-click on a .jar file in the IDE. Choose the ‘Convert to PRC...’ option. A screen will appear with the following options for the PRC which will be generated:

- Main Class

- Short name
- Long name
- Creator code
- Allow networking
- Icon
- Small Icon

The **Main class** should be the first Spotlet which will be run by your application. The **Creator code** may contain a Creator ID which has been registered with Palm, Inc. If it is left blank, the application will appear as a generic application. If the application will be using networking on the Palm OS Platform, the **Allow networking** box should be checked. Note that this will require a network connection when starting the application. If no Bitmaps are selected for the **Icon** or **Small Icon**, then generic icons will be to display the program on the Palm OS Platform. A **Small Icon** should be 15x9, and a **Icon** should be 32x22. If larger bitmaps are used, only the pixels in the upper left hand side would be used.

The Preverifier executable must be set up in the Program Settings (page 32) for this to work.

Load classes onto the Palm OS Platform

Once a class has been converted to a PRC, it is nearly ready to be loaded onto the Palm OS Platform. If the “PRC Install Tool Executable” has been set in the Program Settings, double-clicking the PRC in the IDE will cause it to be loaded onto the Palm OS Platform device the next time a HotSync(TM) operation is performed.

This chapter will discuss the Object Palette and how to use it to assemble Graphical User Interfaces on the Palm OS Platform. It will also cover options of the Palm OS Platform Emulator.

It will discuss

- Assembling parts
- Each of the kjava parts
- The Palm OS Platform Emulator

Assembling A Program Using The Object Palette

When a Composer is opened from the IDE, two additional windows are created: the Object Palette and the Palm OS Platform Emulator.

Object Palette

The Object Palette contains all of the graphical parts which are available to be used in a project. When the user clicks the mouse button over one of the icons in the palette, that icon will appear depressed to indicate that it is selected.



Palm OS Platform Emulator

The Emulator is a live prototype which is updated interactively. The Emulator uses actual Java controls and layouts, and reflects how the application will function and behave on the Palm OS Platform. All of the controls are active and will respond to user interaction. The layout will show the true part placement.

The Emulator initially shows a blank Palm OS Platform screen. Whenever the user clicks the mouse in the Emulator, the part which is selected in the Object Palette will be placed in that space.

Object Palette Pages

The pages in the Object Palette are

- ***kjava*** These parts are the graphical components which make up the *kjava* package.

- **Recycled** This page is the ‘Recycling Bin’ for parts. When the Recycle button in the Composer is pressed, the part which is showing in the part list will be removed from the Emulator and moved to the Recycled page in the Palette. It can then be placed back into the Emulator.

The Object Palette can exist in its own window or it can be placed at the bottom of the Composer window. The user can switch between these two positions by closing the window or by pressing the **Detach** button, respectively.

Kjava Parts

CheckBox

A CheckBox holds a single line of text and a graphical On/Off indicator. The text and the initial state of the CheckBox can be specified in its properties page.

The kjava implementation of CheckBox does not have a getState() method. Users may wish to manually keep track of the state of the CheckBox.

When a CheckBox is pressed, it will change its state and redraw itself. Additional user code may be entered into the penDown page for the CheckBox.

RadioButton

A RadioButton holds a single line of text next to a graphical On/Off indicator. Each RadioButton is also assigned a group. Only one RadioButton in each group can be in the On state at a time.

The text and the initial state of the RadioButton can be specified in its properties page. Pressing the ‘Group Editor’ Button will launch a dialog displaying the current group and allowing the user to add, edit, or remove groups. All RadioButtons are initially set to a group named ‘defaultGroup’. Once the user has created other groups in the Group Editor, these groups can be chosen from the Group choice field.

All changes in the Group Editor effect all of the RadioButtons in that Composer. Once a group is created it is available to all RadioButtons.

Additional user code responding to a `RadioButton`'s `penDown` event may be entered on the `penDown` page.

Button

A `Button` allows the user to indicate when an action should be performed. It can hold a single line of text, or a `Bitmap` image, which can be specified in the properties page.

When a button is pressed, it will flash briefly. Additional user code responding to a `Button`'s `penDown` event may be entered on the `penDown` page.

TextField

A `TextField` is a single line entry field for text input. In its properties page, the user can specify the label of the `TextField`, default text to appear, and the size the Field should be. The `TextField` can be set to change all text entered by hand to upper case.

Simplicity will automatically write code to handle focus management of `TextFields`, causing a `TextField` to gain sole focus when it is tapped. Additional user code responding to a `TextField`'s `penDown` event may be entered on its `penDown` page. A `TextField` also has a `keyDown` page where additional user code responding to a key event may be entered.

TextBox

A `TextBox` is a multi-line label. The size and initial text of the `TextBox` may be set. Note that although `TextBox` will wrap words, it does not scroll text.

Scroll Text Box

A `Scroll Text Box` is a `Text Box` with a scrollbar added.

Note that there is a bug in Sun Microsystem's current implementation of the `ScrollTextBox` class: setting the text using the `setText()` method will cause the scrollbar to disappear. The scrollbar can be forced to reappear by calling the `setBounds` method of the `ScrollTextBox` after setting text.

The default setting for a `Scroll Text Box` is for it to scroll in response to the "page up" and "page down" buttons on a Palm OS Platform based handheld computer. To

make a Scroll Text Box ignore these buttons, uncheck the box titled “Respond to page up/page down keys” on its properties page. Additional user code responding to a Scroll Text Box’s penDown or penMove events may be entered on the penDown and penMove pages.

Select Scroll Text Box

A Select Scroll Text Box is very similar to a Scroll Text Box. It has the added capability to select a single line from the display.

Slider

A Slider can be used to graphically select a value from a range of values by moving a marker.

Additional user code responding to a Slider’s penDown or penMove events may be entered on the Slider’s penDown and penMove pages.

ValueSelector

A ValueSelector is an object which can be used to choose a number by pressing one of three buttons. The three buttons increase the current value, decrease the current value, or randomly set the current value. The maximum and minimum possible values may be set in the property sheet, as well as the starting value. Note that the ValueSelector is designed to display values between -9 and 99 properly. Setting the maximum value to 100 or higher, or setting the value to outside this range may cause errors in the drawing of the ValueSelector.

The Palm OS Platform Emulator

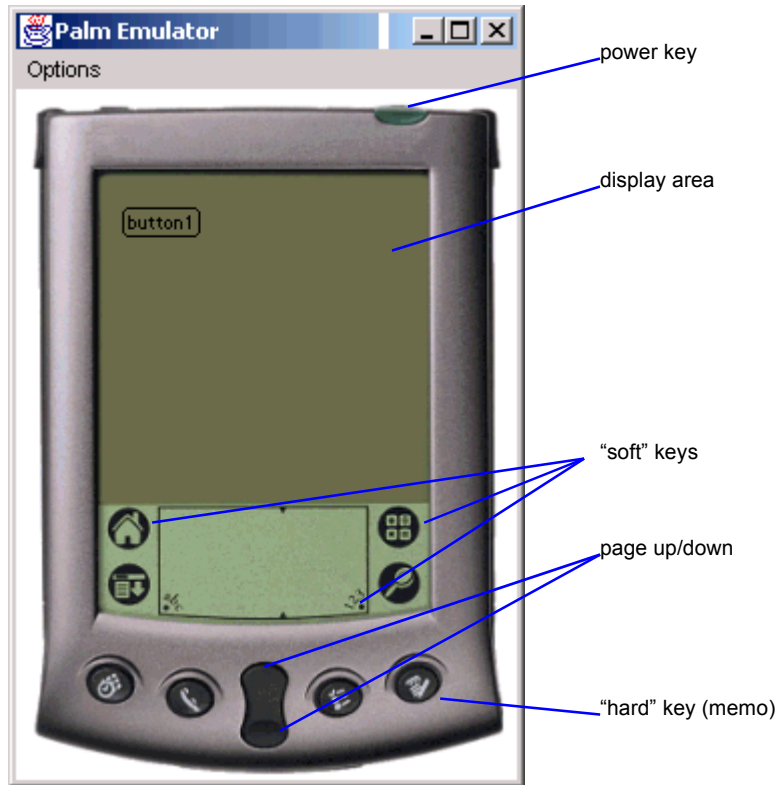
The Emulator is a live prototype of a Spotlet running on the Palm OS Platform. The model will reflect how the application will function and behave. All of the controls are active and will respond to user interaction. The Emulator will show the true part placement. Palm OS Platform database operations are simulated by creating, reading, and writing information to files (located inside the .simplicitypalmos directory) in a way which is identical to the operation of the kjava Database class’s operations on the Palm OS Platform.

The Emulator simulates a sent beam by storing the “beam” in a file named “PalmBeamClipboard” located in the .simplicitypalmos directory. The Beam Clipboard Editor will appear after each beam. The Clipboard Editor is a very simple text editor which can make changes to the text in the PalmBeamClipboard file. If the beam is not text, the Clipboard Editor may appear blank, but the data will be written in the PalmBeamClipboard. The Emulator simulates a received beam when ***Send Beam Event*** is selected from the ***Options*** menu in the Emulator.

Note that on a Palm OS Platform based handheld computer, a window from the Palm OS Platform will appear whenever a beam is sent or received. This may clear part of the screen, so a repaint is suggested after sending or receiving a beam.

The Emulator will beep when it is asked to play a sound.

In some circumstances, a warning sign will appear in the upper left-hand corner of the Emulator. Pressing this button will cause the warning to be displayed in the standard output.



Emulator Options

The Emulator has an options menu which has the following items on it

- ***View double size*** This will show the Emulator at double size. This can be useful depending upon the resolution and settings of the monitor.
- ***Send Beam Event*** This will cause the Emulator to read in the contents of the PalmBeamClipboard file as a new beam being received.
- ***Edit Beam Clipboard*** This will open the Beam Clipboard Editor.

Building Spotlets

The Emulator initially consists of an empty screen. Parts are added to the Emulator by choosing them from the Object Palette and clicking them onto the Emulator using the mouse. Once a part has been put into the Emulator it is live and can respond to user interaction. The user can specify what code a part will execute in response to an event, by entering the code in the appropriate area of a part's properties notebook.

Clicking any part with the mouse will also cause its properties notebook to appear in the Composer window. Some parts may be difficult to click with the mouse. The properties notebooks for those parts can be viewed by choosing the part name from the part list at the top of the Composer window.

This chapter will discuss the Code Sourcerer. The Code Sourcerer will generate Java statements based upon user choices.

This chapter will cover

- How to use the Code Sourcerer
- The major categories in the Code Sourcerer.

Using the Code Sourcerer

Every code area in the Composer has a Code Sourcerer button. When the user presses this button, a dialog will appear which will guide the user through a context-sensitive set of pages. By making a few selections, and filling in a few entry fields the user can construct Java statements to perform a wide variety of activities.

The Code Sourcerer presents the user with a series of panels. Once finished with a panel, the user can proceed to the next panel by pressing the ***Next*** button. The user can return to a previous panel using the ***Back*** button. At any time the user can quit the Code Sourcerer by pressing the ***Cancel*** button. Once the Code Sourcerer has obtained enough information to completely write the requested code, the ***Done***

button will become enabled. Pressing **Done** will write the code. Each of these buttons will become enabled/disabled in a context-sensitive manner. In some cases, both **Next** and **Done** will be disabled. This indicates that there are still critical fields to be filled in before the Code Sourcerer can proceed.

Each time the Code Sourcerer is invoked, the code which is produced is added to the code area. This code can be used as is, combined with other user code, or edited to meet particular needs. The code will be appended to the end of the code area unless the control key is held down when the Code Sourcerer is pressed.

The first panel of the Code Sourcerer contains ten categories. The user can choose from

- **Change a property of an existing part...** Choose a part and then modify any attribute of the part.
- **Ask a part about one of its properties...** Choose a part and then retrieve the value of any attribute.
- **Palm OS Platform Database operations...** Interact with the Palm OS Platform database, which is used to store information on Palm OS Platform devices.
- **Palm OS Platform Beaming operations...** Send messages to another machine via the built-in infrared port of a Palm OS Platform based handheld computer.
- **Palm OS Platform Screen Painting...** Draw lines and other shapes on the screen, redraw a part or parts, clear the screen.
- **Palm OS Platform Network operations...** Get the contents of a web page or connect to another machine over the internet.
- **Palm OS Platform Spotlet operations...** Show help text, switch to another spotlet, respond to different buttons.
- **Perform floating point computations...** Obtain the results of a floating point computation.
- **Declare a new variable...** Create a new variable of any type and optionally initialize it.
- **Miscellaneous...** Play a sound, get the ID of your machine, more...
- **Java system operations...** Interact with the Java Virtual Machine or the Palm OS Platform.
- **Java Language Statements...** Conditional statements (if, if else) and loop statements.

Change a property of an existing part

This option presents the user with a list of the parts which have been added to the Palm OS Platform Emulator. The user should choose the part that they wish to modify and press the *Next* button. The panel which follows will let the user choose from among the specialized properties for that part.

If it is appropriate, Simplicity will generate code which will cause all parts to be repainted after the part has changed.

Ask a part about one of its properties

This option presents the user with a list of the parts which have been added to the Palm OS Platform Emulator. The user should choose the part that they wish to query and press the *Next* button. The panel which follows will let the user choose from among the specialized properties for that part.

After the user has chosen the property that they wish to retrieve, a dialog will ask where the value should be stored. The input field will contain a suggestion which will create a new local variable. While the variable type must remain the same, the name may be changed to any unused, legal Java name. If the user would like to store the value in a variable that has been previously declared, the entire contents of this field can be changed to that name.

Palm OS Platform database operations

This option allows the user to create code which interacts with the Palm OS Platform database, which is the method that the Palm OS Platform uses to store data on the Palm OS Platform. The options include

- Open a database...
- Create a database object...
- Check to see if a database is open...
- Close a database...
- Add a record to a database...
- Set a record in a database...
- Read a record from a database...
- Delete a record from a database...
- Get the number of records in a database...

Before the Spotlet can interact with the underlying Palm OS Platform database, you must ***create a database object***. The information needed to do this includes the Creator ID and Type ID of the database. Your Creator ID should be registered with Palm, Inc. at <http://www.palmos.com/dev/tech/palmos/creatorid/reg.html>.

Use ***create a database object*** if you wish to define the database object in the Declarations area of your code but not actually open the database until the constructor. Before actually using the database object, you will want to use the ***open a database*** option. This will ensure that the database is usable, because the code which the Code Sourcerer generates will create a new database if it does not already exist. If you will be using a database object inside a limited scope (for example, just inside a particular method), you can make a new database object and then open it.

It is possible to cause serious errors in the Spotlet if an attempt is made to access a database which is not already open, so a ***check to see if a database is open*** is a very useful thing to do before attempting to do other database operations.

Although databases can be closed by the Spotlet, it is important to ***close databases*** manually when they are no longer being used.

The Sourcerer also has options to ***add a record*** to a database, which will add a new record to the end of a database. You can also ***set a record*** of a database, and ***read a record*** or all the records from the database. You can ***delete a record*** or all the records from the database. Finally, there is an option to ***get the number of records*** currently in a database.

Palm OS Platform Beaming operations

Palm OS Platform based handheld computers can “beam” information by using their built-in infrared ports. The Code Sourcerer will generate code which will beam information to another Palm OS Platform based handheld computer. The options are

- Beam text...
- Beam a byte array of a specified name

Note that a Spotlet receives “beamed” information as a byte array in a special method, beamReceived.

Palm OS Platform Screen Painting

This option lets the Sourcerer draw various graphics objects on the screen. They include

- A line
- A filled rectangle
- A filled rounded rectangle
- A border
- A text string

Each object will require a location as well as some additional information to draw it on the screen. All of these parts have an option for the draw mode. For the line, filled rectangle, filled rounded rectangle and border, the “gray” draw mode produces dotted graphics, and the “invert” mode will draw black on green and green on black. For text strings, the “gray” draw mode is identical to plain text, and the “invert” mode always draws green text on a black background.

In addition, there are several other possible drawing operations:

- Clear the screen
- Set clipping region
- Reset clipping region
- Paint an existing part
- Paint all parts

When the screen is cleared, no parts are removed from the program, but they will become invisible until the next time they are painted. A user may choose to make the Palm OS Platform *paint a part* in an application. This is important, because in kjava, parts are not always redrawn when they are changed programatically. The option to *paint all parts* may also be available, if it is appropriate.

Palm OS Platform Network operations

The Sourcerer can generate code which will let the Palm OS Platform connect to other computers via HTTP or through a socket connection. The options are

- *Make an HTTP connection...*
- *Make a socket connection*
- *Perform a Socket operation...*

- ***Close an HTTP connection***
- ***Close a socket connection***

To make an HTTP connection or a socket connection, a URL must be entered. Note that the URL for socket connections must include a port number. There are six options for an HTTP connection. They are

- Get the entire contents of the URL
- Get the encoding of the page
- Get the length of the page
- Get the type of the page
- Open an InputStream
- Open an OutputStream

There are similar operations for sockets:

- Send a String
- Receive a String
- Open an InputStream
- Open an OutputStream

Note that it is important to close HTTP and socket connections, as well as any InputStreams and OutputStreams that may have been opened.

Palm OS Platform Spotlet operations

There are several different Spotlet operations possible. They are

- ***Display help text...*** Display a screen of help text from a Spotlet.
- ***Switch to another Spotlet...*** Close the current Spotlet and start another.
- ***Make Spotlet respond to different events...*** The Spotlet can either respond to all buttons pressed or just the page up/page down buttons. For more information on events, see “Event Options” on page 50.

Perform floating point computations

Java on the Palm OS Platform does not directly support floating point calculations. This option will let you enter a calculation using the `com.datarp.calc.Calculator` class which will evaluate floating point calculations in a String. For example, the

String “5/2” would return a new String “2.5”. For further information on the `com.datarp.calc.Calculator` class, users may read the javadoc documentation which addresses issues such as precision and reproducibility of the results from this class.

The Sourcerer will ask you for the text of the calculation, which can be entered as a string, as the string from another object in the project, or as a variable expression.

Declare a new variable

This option lets the user create a new variable. The user must enter a name for the new variable. The user can choose a primitive type or a class type for the variable. If a class type is chosen, the name for the class should be entered. The full class name should be specified unless the class is a member of the following packages which are imported by default.

- `java.lang.*`
- `java.util.*`
- `java.io.*`
- `javax.microedition.io.*`
- `com.sun.kjava.*`

Any array dimensionality and set of modifiers can be chosen. The accessibility radio buttons must be set to **default** for all code areas except the ‘**Declarations**’ code area.

When the **Next** button is pressed, the user will be able to indicate how the variable should be initialized. For primitive types, the user should enter a value. For class types, the user will be given a list of the available constructors as well as the **null** value. If a constructor is chosen, the next panel will contain fields for the parameters.

Java system operations

These options interact with the Java Virtual Machine and the underlying Palm OS Platform. They include

- Query the total amount of system memory
- Query the amount of free memory
- Run the garbage collector
- Get a string with the current date and time

- Suspend the current thread for a specified number of milliseconds
- Exit the program with a termination code
- Write text to the standard output...
- Write text to the standard error...

Note that the standard way to exit a Spotlet is to exit with termination code 0.

Miscellaneous

This option includes a variety of operations. They include

- ***Play a system sound...*** You can choose from the different sounds which the Spotlet can play on the Palm OS Platform.
- ***Get the FlashID*** This corresponds to the 12-character serial number of your Palm OS Platform based handheld computer; in the Emulator the string “SimplicityPP” will be returned.
- ***Convert from Strings to other types...*** Convert text from a String to other Java objects or primitives, including a byte array.
- ***Convert variables to Strings...*** Convert a variable or object (including a byte array) into a String.

Java Language statements

This option lets the Sourcerer create Java statements.

Conditional Statements

Conditional statements let the user specify code which will only happen if a certain condition is met. There are two types of conditional statements,

- if...
- if else...

In both cases, the User must enter the statements which happen if the condition is true. A comment shows where this is appropriate.

Loop Statements

Loop statements let the Java program repeat certain portions of code in a controlled manner. There are several different types of loop statements available from the Sourcerer. Each one has its own advantages.

- ***do...*** this will run the statements inside the loop, then check to see if the loop condition is true. If it is true, then the loop will be run again and the loop condition tested again. This means that the loop will be run at least once.
- ***while...*** this will check to see if the loop condition is true. If it is true, then the loop will be run, and the loop condition tested again. This means that it is possible that the loop will not be run at all.
- ***counter...*** this will start a loop which is connected to a counter. The counter will start at a starting number, end at an ending number, and count in user-defined multiples. This is a quick way to make a loop run a specific number of times, or to have access to a variable which is counting up or down.

The Java Command Window allows the user to experiment with Java language statements, execute methods dynamically for testing and debugging, and view the local symbol tables while working in a Composer.

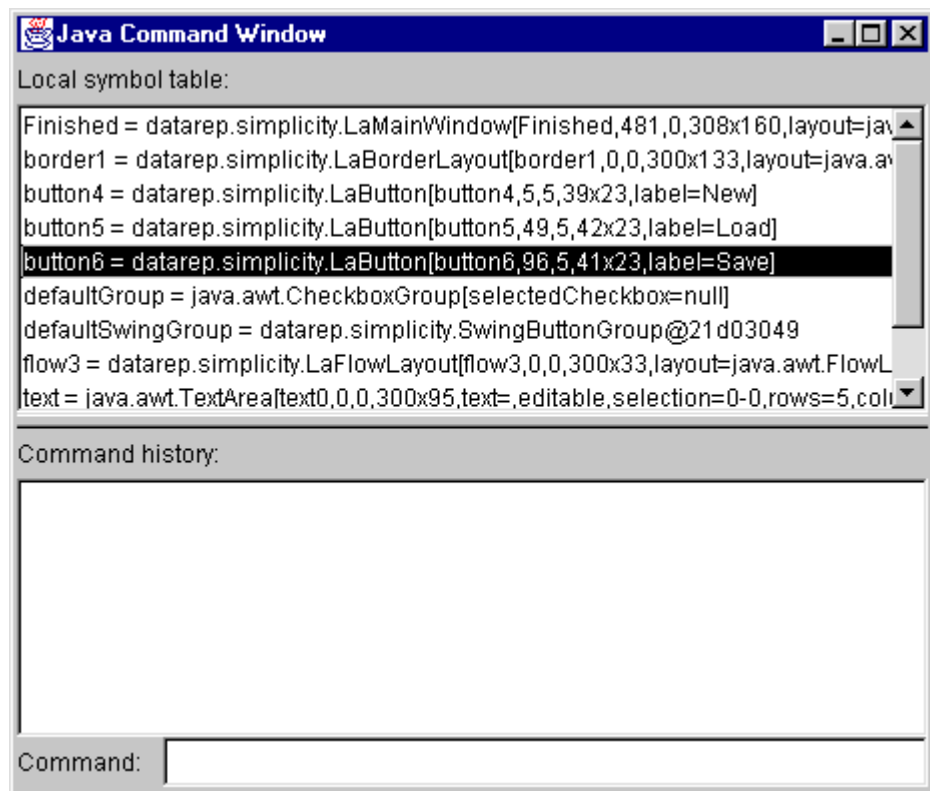
This chapter will discuss:

- Inputting Java statements
- Reviewing the command history
- Manipulating the local symbol table
- Using the Java Command Window from the IDE
- Using the Java Command Window from the Composer
- Using the Java Command Window from the Debugger

Using The Java Command Window

The Java Command Window allows the user to execute individual Java statements to test code, manipulate running applications, and discover information about running applications.

The Window itself three major parts: The Command Input line, The Local Symbol Table, and the Command History.



Command Input

The Command Input, which appears at the bottom of the Command Window, allows the user to enter one or more Java language statements. These will be executed when the user presses the Enter key.

It is not necessary to terminate a single statement with a semicolon, as one will be added implicitly. Multiple statements may be separated by semicolons. For example, the following line may be entered into the Command Input:

```
import java.util.*;  
Vector v = new Vector();  
int i=0; while(i<8) v.addElement(String.valueOf(i++));  
System.out.println(v);
```

Local Symbol Table

The Local Symbol Table contains a list of all of the local identifier names as well as the String value of the object being referenced.

Double clicking a symbol will add the name of the symbol to the Command Input field and then launch the Sourcerer's Apprentice (page 97) to allow the user to select a method or field name from the class.

Command History

The Command History lists all of the commands which you have previously entered. Selecting any command will copy the command to the Command Input field for further editing or execution. Double-clicking a command will immediately re-execute the command.

The Three Java Command Window Contexts

The Java Command Window is used in different ways depending on the context from where it has been launched.

IDE

A Command Window may be launched from the IDE, using the “New Command Window...” item on the “Project” menu. This Command Window will inherit the classpath as set in the dynamic classpath in the IDE. Its symbol table will initially be empty.

This Command Window may be used for general purpose experimentation with Java statements and for execution and testing of classes.

Composer

A Command Window may be launched from any Composer, using the “Command Window” item on the “Program” menu. This Command Window will inherit the classpath as set in the dynamic classpath in the IDE. Its symbol table will contain all of the symbols in the class scope of the Composer. For example, a Command Window opened from within a Frame Composer will contain references to the Frame itself, all of the graphical components that have been added to the Frame, and any declarations added to the Declarations code page.

This Command Window may be used for testing the class being constructed in the Composer. Properties of the components in the Composer may be changed programmatically for testing and experimentation purposes. Any method in the Methods code area may be executed for testing.

Debugger

A Command Window is launched whenever the Debugger is started. This Command Window allows the user to manipulate the application being debugged directly from within its own Java Virtual Machine. The Symbol table is initially empty.

This Command Window has two additional buttons at the top that are specific to the debugger. The first, ***Run program...***, allows the user to specify a class with a `public static void main(String[])` method for execution. The second, ***Load classes...***, lets the user specify a set of classes to load into memory.

This Command Window may be used in many ways depending on the type of debugging being done. It may be used to create specialized invocations of Java class methods for testing and debugging, without the need to have a main application previously created to invoke them. It also may be used to query and modify running applications through static method calls.

This chapter will discuss Simplicity's Debugger. It will cover

- Starting the Debugger
- The Debugger window
- The Command Window

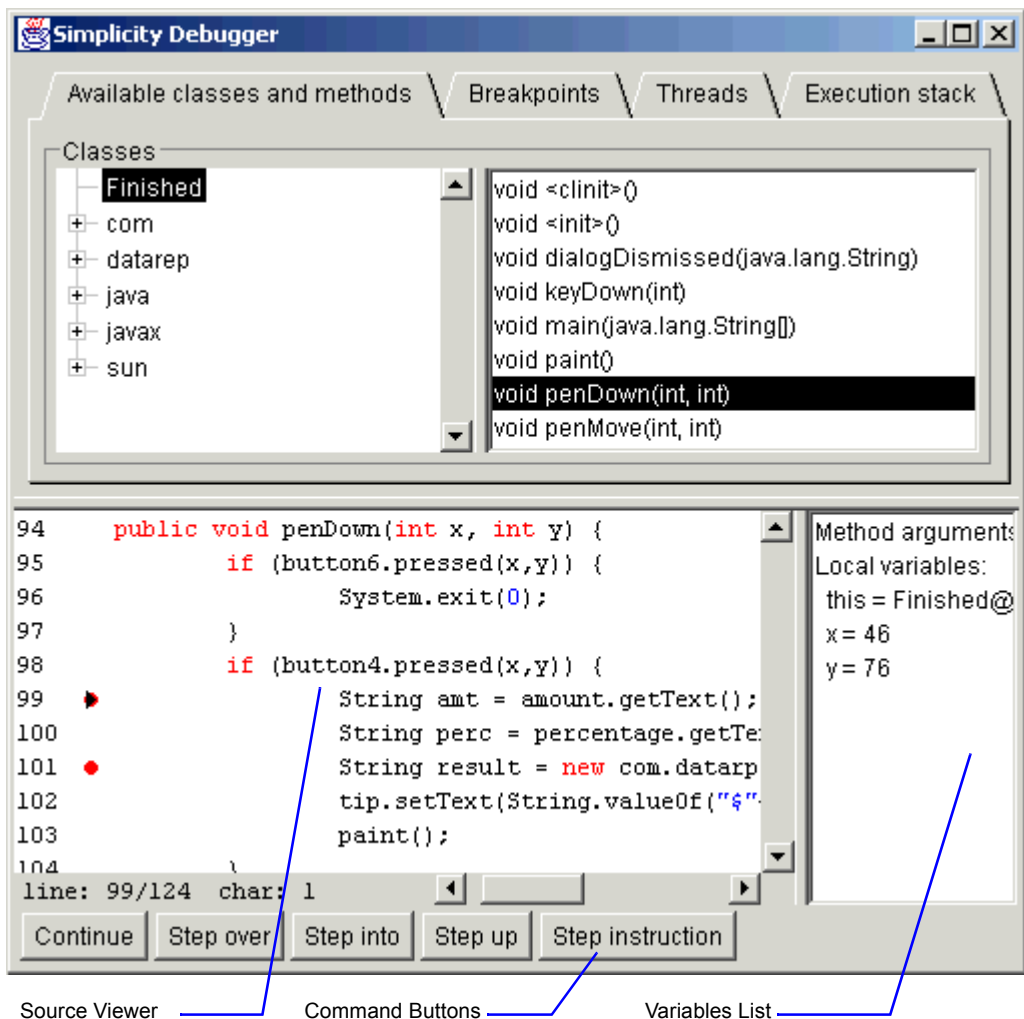
Starting the Debugger

The debugger is started from the Simplicity IDE. When you right-click (or Ctrl-click for single button mice) on any Java Source or Java Class icon, the pop-up menu that appears will include options to launch the debugger. Initially, the Java class which is selected in the IDE will be loaded into the debugger.

Once the Debugger has started, four windows will appear. They are the Debugger Window, the Debugger Command Window, the Palm Emulator window, and the Run Class dialog. The Run Class dialog will allow you to start a Spotlet which contains a `public static void main (String[])` method.

The Debugger Window

The debugger window is the central location for controlling and debugging your application. From the Debugger window, you can view all of your classes and their methods, set breakpoints, observe and modify thread states, observe the execution stack, view the source code being debugged (along with markers indicating breakpoints and current location), and view all local variables.



The top portion of the Debugger window contains a tabbed pane with four tabs labeled “Available classes and methods”, “Breakpoints”, “Threads”, and “Execution Stack”. The information displayed on each page is updated in response to program events. An update may be forced by clicking on a tab at the top of a page. (This may be used to refresh the current page, or any other page.)

Available classes and methods

This page contains a tree showing all of the classes that have been loaded into the debugger’s Virtual Machine.

Selecting any class in the tree will display a list of the class’ methods in the list to the right. If available, it will also display the class’ source code and local variables in the Source Viewer and Variables List.

Double clicking any method in the methods list will set a breakpoint at the entrypoint to that method.

Breakpoints

The Breakpoints tab displays a list of all of the current breakpoints. The name of the class and the line number are listed.

Selecting any of the breakpoints in the list will display that source file in the Source Viewer, and jump to that line.

Double-clicking a breakpoint will clear the breakpoint.

Threads

The Threads tab displays all of the threads that the application being debugged contains, as well as information about the current state of each thread.

The four buttons at the bottom of this page can be used to individually suspend or resume a particular thread, or to suspend or resume all threads simultaneously. Note that these buttons have no effect while an application is stopped at a breakpoint. You must press the ***Continue*** button to continue execution from a breakpoint.

Selecting a thread makes that thread the active thread being debugged. The Execution Stack, Source Viewer, and Variables list will display information in the context of the active thread.

Execution Stack

The Execution Stack page displays the execution stack and current location within the current thread's stack frame. This only has meaning when the thread is suspended or is stopped at a breakpoint.

Selecting a method call within the execution stack will cause the Source Viewer and Variables list to display their contents within the context of that method.

Source Viewer

The Source Viewer displays the source code (if available) for the class being debugged. It is context sensitive, and will jump to the source file or method or line number being chosen in the class list, breakpoints list or execution stack.

Double clicking a line will set a breakpoint at that line or clear a breakpoint if one already exists. A breakpoint can only be set at a line containing an executable statement.

Variables List

The Variables List contains a list of all local variables and method arguments in the current Source Viewer context. It also contains a reference to the “this” pointer in the current class context.

Double clicking any item in the Variables List will launch a window containing an additional Variables List showing an expanded view of that item. Double-clicking an item in the new list will again launch an additional Variables List. Class fields may be viewed to an arbitrary depth in this manner. The contents of any given Variables List window may be unavailable depending on the current active thread and stack depth, but will be refreshed when the appropriate context becomes active.

Command Buttons

At the bottom of the Debugger Window are five Command Buttons, which are used to control execution when an application is stopped at a breakpoint. The buttons are:

- ***Continue*** This button will resume normal execution of a Spotlet after it has been stopped at a breakpoint.
- ***Step over*** This button will execute the line of code at the cursor, and attempt to remain in the same stack location.

- **Step into** This button will begin execution of any method call in the line of code at the cursor, and will leave the cursor at the top of that method call for further debugging.
- **Step up** This button will instruct the debugger to finish executing the method in the current stack location, and leave the cursor at the next line in the stack location above the current stack location.
- **Step instruction** This button will execute a single Java byte code instruction within the current line of code.

Java Command Window

A Java Command Window is launched whenever the Debugger is started. The Command Window allows the user to manipulate the application(s) being debugged directly from within its own Java Virtual Machine. The Command Window may be used in many ways depending on the type of debugging being done. It may be used to create specialized invocations of Java class methods for testing and debugging, without the need to have a main application previously created to invoke them. It also may be used to query and modify running applications through static method calls.

This Command Window has two buttons at the top that are specific to the debugger. The first, **Run program...**, allows the user to specify a class with a `public static void main(String[])` method for execution. The second, **Load classes...**, lets the user specify a set of classes to load into memory.

Run program

The **Run program** dialog lets the user specify a class with a `public static void main(String[])` method for execution. The user may also enter command line parameters (one per line) into the dialog. The specified class's `main` method will be executed when the **Run** button is pressed. All applications started through the **Run program** dialog execute within the same Virtual Machine and are capable of interacting with each other through static methods.

Load classes

The **Load classes** dialog allows the user to instruct the Java Virtual Machine to load particular classes into memory. It is primarily used to force the debugger to load a class so that the class can be viewed in the Debugger Window or so that breakpoints can be set in the class.

A classpath tree is on the left panel. The middle panel shows a list of classes in the selected package. Selecting and pressing the *Add* button will add the classes to the panel on the left. The classes are loaded when the *Ok* button is pressed.

Advanced Features - Extending the IDE

This chapter will discuss how to extend Simplicity's Integrated Design Environment (IDE) and how it can be customized.

It will cover

- How to extend the IDE
- Some Samples of the extended IDE

Extending the IDE

The menubar in the Simplicity IDE can be customized to meet the user's needs and to allow for easy access to programming utilities.

The IDEmenu.config file

When Simplicity starts, it looks for a file called IDEmenu.config in the Personal Settings Directory (See page 6). If this file does not exist, Simplicity will use the default factory settings for the menus.

A copy of IDEmenu.config is included in the install, but it will not be used unless it is moved to the Personal Settings Directory.

The IDEmenu.config file is a plain text file, which uses a simple markup language similar to HTML to format the menus. Capitalization of the tags and commands are also not important. Spaces and line breaks between tags are not important. Like HTML, tags must be contained within angled brackets `<>`.

The only allowed tags are

- `<menubar>` `</menubar>`
- `<menu>` `</menu>`
- `<menuitem>` `</menuitem>`
- `<separator>`
- `<action>` `</action>`
- A `#` sign signifies a comment, which causes the rest of the line to be ignored.

MenuBar

The IDEmenu.config file must have both the `<menubar>` and `</menubar>` tags. The text between these will be used to build the menus.

Menu

The Menu tag starts a menu. The ***label=*** command gives the name. Quotation marks are not necessary if the name is only one word; names longer than a word should be enclosed in straight quotes. Menus can be nested (to give sub-menus). Each `<menu>` tag must be paired with a `</menu>`. The following example will create one menu, named File, which only contains a sub-menu named “Open This...”.

```
<menubar>
<menu label=File>
<menu label="Open This..."> </menu>
</menu> </menubar>
```

MenuItem

The MenuItem tag defines a menu item. A MenuItem should be given a name using the ***label=*** command. Quotation marks are not necessary if the name is only one word; names longer than a word should be enclosed in straight quotes. If a MenuItem should be available even when there is no open Project, then the command ***alwaysEnabled=true*** should be included in the MenuItem tag. If this is

not included, the default of *alwaysEnabled=false* is used. For a MenuItem to have any function, it must contain at least one Action tag. The `</MenuItem>` tag is optional, since MenuItems cannot nest. The following sequence will be correctly interpreted by Simplicity as being two MenuItems in a menu: `<Menu label=a>
<MenuItem label=one> <MenuItem label=two> </Menu>`

Separator

The Separator tag creates a separator between MenuItems. In most operating systems, the separator appears as a line.

Action

The Action tag tells Simplicity what to do when a menu item is chosen. It should be enclosed within MenuItem tags. There are two different ways to tell Simplicity what should be done. The *command=* command or the *class=* command can be used.

The *command=* command lets the user specify a command just as would normally be done from the command line of the operating system being used. Tabs can be used to separate parts of the command. The following command might be used to open Netscape on a machine running Windows:

```
command=C:/Netscape.exe
```

Windows normally uses the “\” character to separate directories. However, this character is used to denote escape sequences in Java. To use \s in the above example, the command would have to be `command=C:\\Netscape.exe`.

Even though the MacOS does not have a command line, *command=* will work to open programs with the MacOS. See the next section for an example.

The *class=* command can be used to have Simplicity open a class. The class to be opened must extend `datarep.ide.config.Action`. The `actionPerformed(ActionEvent)` method will be called when the menu item is chosen. There are two possible ways to use this command. One would be to create a new class which extends `Action`; the other would be to extend one of Simplicity’s existing menu commands by extending that class. Examples of both will be given in the next section.

Any number of Action tags can be associated with a menuItem; they will be executed in sequential order. The `</Action>` tag is optional, because Action tags do not nest. Simplicity interprets the following sequence correctly as being two

```
Actions associated with the menuItem "test": <menuItem label=test>
<action command=command1> <action command=command2>
</menuItem>
```

Samples of the extended IDE

The following samples have been shortened, but will demonstrate the proper use of the tags and commands available to extend the IDE. Package names have been omitted from the Java code, but may be used if desired.

Adding a command to the help menu

This example shows how to add a new command to the help menu, using **command=**. The command being added opens up Netscape to the Simplicity home page. The path given is for an iMac; you will have to change this to reflect your computer

```
<Menubar>
# all text from the '#' onwards is a comment.

# ... other menus should be defined here...
<Menu label=Help>
<MenuItem label="User Guide..." alwaysEnabled=true>
# the user guide should always be available
<action class=datarep.ide.config.LaunchUserGuideAction>
# the above action launches the user guide
</menuItem> # this tag is optional
<Separator>
<MenuItem label="Go to data representation's home page"
alwaysEnabled=true>
<action command="/myiMac/Internet/Netscape NavigatorTM
    http://www.datarepresentations.com">
# note that there is a tab separating the path name from
# the web site.
</menuItem>
</menu> # this tag is not optional
</menubar>
```

Adding a new action

This brief example adds a new menuItem called “beep”, which beeps, using the **class=** command. First a new class is created, then the addition to the IDEmenu.config file is described.

First a new BeepAction class must be created and compiled:

```
import java.awt.event.*;
import datarep.ide.config.Action;
public class BeepAction extends Action {
    public void actionPerformed(ActionEvent event) {
        getToolkit().beep();
    }
}
```

Next the following addition needs to be made to the IDEmenu.config file. Note that the class which is used in the action tag is the new BeepAction class.

```
<menuItem label=beep> <action class=BeepAction> </menuItem>
```

Modifying existing actions

In addition to creating new classes extending the Action class, users can easily change existing classes by extending them. In this example, the Refresh menu item (under the Edit menu) will be modified. First a new class will be created which extends RefreshAction, then the modifications to the IDEmenu.config file will be shown.

First the BeepAndRefreshAction class must be created and compiled:

```
import java.awt.event.*;
import datarep.ide.config.RefreshAction;
public class BeepAndRefreshAction extends RefreshAction {
    public void actionPerformed(ActionEvent event) {
        getToolkit().beep();
        super.actionPerformed(event);
    }
}
```

Next the following change must be made to the IDEmenu.config file. The tag which reads

```
<action class=datarep.ide.config.RefreshAction>
```

should be replaced with

```
<action class=BeepAndRefreshAction>
```

Now whenever the Refresh item is chosen from the Edit menu, the user will hear a beep before the refresh occurs. Although this example was simple, more complex classes could be created.

A Complex Action

This last example will show how multiple action tags can be associated with a MenuItem. It will also involve making the Refresh menu item beep, both before and after the Refresh occurs. To do this,

1. Make sure that the BeepAction class (page 87) is compiled.
2. replace the code for the Refresh MenuItem with the following code:

```
<MenuItem label=Refresh> <action class=BeepAction>  
<action class=datarep.ide.config.RefreshAction>  
<action class=BeepAction>
```

More Action tags could be chained together if desired. Also **command=** and **class=** Action tags can be associated with the same MenuItem.

Index

Symbols

.lic_txt 6
.license directory 6

A

AIX 3
array 69

B

beam 66
Beam Clipboard Editor 60
Beaming operations 66
beamReceive 52
Bitmap 26, 30, 41, 58
Bitmap Editor 26, 41
Border Layout 57
breakpoint 80
Breakpoints 79
Button 42, 58

C

Calculator 68
Checkbox 57
Choice 58
class type 69
Class Viewer 26
CLASSPATH 4
Classpath 4, 22, 23
CLDC 23
clear the screen 51, 67
code area 63
Code menu 48, 52
Code Sourcerer 50, 63
Color 38
Command History 74
Command Input line 74
Compile 52
Composer 26, 45, 47, 49, 51, 55, 63, 73, 76
Conditional statements 70
Console 28
constructor code 48, 51
Continue 80
Convert 70
Convert to PRC 53
counter 71
Creator ID 54, 66

D

Data File 30
Data Representations, Inc. i
Debugger 31, 76, 77
debugging 73
declaration code 48, 51
Detach 57
dialogDismissed 52
Directories 31
do loop 71

E

Edit locations using mouse 51
Editor 35
Empty The Recycling Bin 47, 49
Emulator 46, 49, 56, 59
Enable Events 49
Exclude packages 53
Execution Stack 79
External Editors 31, 32

F

Feedback 7
FlashID 70
floating point 68
Flyer 59
Flyouts 32
Folders 22
Folders area 23

G

garbage collector 69
Generate code 48
Graphics 41
Group Editor 24, 29
Groups 23

H

HotSync KVM 28
HTML 27, 30
HTTP 67

I

IDE 21, 22, 46, 75
IDEmenu 6, 83
IDEmenu.config 83
import 50
Indentation 37
Initialize Class 48, 49

Installation 1
Installation Directory 31
Integrated Design Environment 21, 73, 77, 83
IRIX 3

J

J2ME 52
J2ME Install Wizard 28, 32
Jar 53
Java class file 26
Java Command Window 30, 48, 73, 81
Java Compiler 31
Java Editor 26, 31, 32, 33
Java File 30
Java source file 26
Java system operations 69
Java Virtual Machine 1, 69
JDB 31
JDK 1

K

keyboard shortcuts 37
keyDown 50, 52
kjava 23

L

Label 57
Layouts 56, 57
Linux 3, 4, 5, 6
Listbox 58
Local Symbol Table 74
Loop statements 71

M

Mac OS 2, 3, 5
Main App 29, 51
memory 69
method code 48, 51

N

Network operations 67

O

Object Name 49
Object Palette 31, 32, 46, 55, 56, 62
OS/2 Warp 3, 4

P

- package 23
- paint 52
- Paint a part 67
- Palm OS Platform database 65
- Palm OS Platform Emulator (see Emulator) 56
- PalmBeamClipboard file 60
- Part List 47
- pen events 51
- penDown 50, 51
- penMove 50
- penMoved 51
- penUp 51
- Perl 5 38
- Personal 31
- Personal Settings 31
- Personal Settings Directory 28, 59
- PRC 53
- Pre-verifier 32
- PreVerify 53
- primitive type 69
- Printing 31, 33, 35, 38
- program 6
- Program menu 48, 49
- Program Settings 27, 28, 31
- Progress Bar 59
- Project Groups 23
- Project Tree 22
- properties 65
- Property Notebooks 49

R

- Radio Button 57
- Recycle Current Part 47
- Recycle current part 49
- Recycled 57
- Reset clipping region 67

S

- SCO 3
- Screen Painting 70
- Search and Replace 35, 38
- Set clipping region 67
- socket 67
- Solaris 3, 4
- sound 60, 70
- Sourcerer's Apprentice 35, 36, 39, 75
- Spotlet 46
- Starting Points 53

stdin/stdout/stderr 70
Step instruction 81
Step into 81
Step over 80
Step up 81
System Requirements 1, 2

T

Technical 7
Text Area 59
Text Editor 27, 32
Text Field 58
Text File 30
thread 70
Threads 79

U

Unix 3, 4
UnixWare 3

V

variable 69
Variables List 80
View Code 48
View Dependencies 53

W

web browser 27, 32
while loop 71
Window operations 69
Windows 2
Windows 95 3, 7
Windows NT 4, 7