

Pro J2ME Polish

Open Source Wireless
Java Tools Suite

ROBERT VIRKUS

Pro J2ME Polish: Open Source Wireless Java Tools Suite

Copyright © 2005 by Robert Virkus

Lead Editor: Steve Anglin

Technical Reviewer: Thomas Kraft

Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis,

Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, Jim Sumser

Associate Publisher: Grace Wong

Project Manager: Beth Christmas

Copy Edit Manager: Nicole LeClerc

Copy Editors: Marilyn Smith, Kim Wimpsett, Nicole LeClerc

Assistant Production Director: Kari Brooks-Copony

Production Editor: Ellie Fountain

Compositor: Dina Quan

Proofreader: Linda Seifert

Indexer: John Collin

Artist: Kinetic Publishing Services, LLC

Cover Designer: Kurt Krames

Manufacturing Manager: Tom Debolski

Library of Congress Cataloging-in-Publication Data

Virkus, Robert, 1949-

Pro J2ME Polish : open source wireless Java tools suite / Robert Virkus.

p. cm.

Includes index.

ISBN 1-59059-503-3 (hardcover : alk. paper)

1. Java (Computer program language) 2. Wireless communication systems--Programming. I. Title.

QA76.73.J38V57 2005

005.13'3--dc22

2005016571

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013, and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders@springer-ny.com, or visit <http://www.springer-ny.com>. Outside the United States: fax +49 6221 345229, e-mail orders@springer.de, or visit <http://www.springer.de>.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit <http://www.apress.com>.

The information in this book is distributed on an "as is" basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at <http://www.apress.com> in the Downloads section.

*This book is dedicated to the people who fight against
software patents and for a better world.*



Contents at a Glance

Foreword	xvi
About the Author	xviii
About the Technical Reviewer	xix
Acknowledgments	xx
Introduction	xxi

PART 1 ■ ■ ■ Getting Ready

■ CHAPTER 1	Quick Setup Guide	3
■ CHAPTER 2	Installing the Prerequisites	7
■ CHAPTER 3	Installing J2ME Polish	11
■ CHAPTER 4	Integrating J2ME Polish into IDEs	19

PART 2 ■ ■ ■ Using J2ME Polish

■ CHAPTER 5	Getting to Know J2ME Polish	29
■ CHAPTER 6	The Device Database	39
■ CHAPTER 7	Building Applications	55
■ CHAPTER 8	Preprocessing	101
■ CHAPTER 9	The Logging Framework	123
■ CHAPTER 10	Using the Utilities	133
■ CHAPTER 11	Game Programming with J2ME Polish	145
■ CHAPTER 12	Working with the GUI	159
■ CHAPTER 13	Extending J2ME Polish	231

PART 3 ■ ■ ■ Programming in the Real World

■ CHAPTER 14	Overview of the Wireless Market	273
■ CHAPTER 15	Dancing Around Device Limitations	283
■ CHAPTER 16	Optimizing Applications	325

PART 4 ■ ■ ■ Appendix

■ APPENDIX	351
■ INDEX	419

Contents

Foreword	xvi
About the Author	xviii
About the Technical Reviewer	xix
Acknowledgments	xx
Introduction	xxi

PART 1 ■ ■ ■ Getting Ready

■ CHAPTER 1	Quick Setup Guide	3
	Installing J2ME Polish	4
	Launching the Sample Applications	4
	Exploring the Sample Applications	5
	Summary	6
■ CHAPTER 2	Installing the Prerequisites	7
	The Java 2 SDK	7
	The Wireless Toolkit	8
	WTK Versions	8
	WTK for Mac OS X	8
	IDE	8
	Ant	9
	Vendor-Specific Emulators	9
	Summary	10
■ CHAPTER 3	Installing J2ME Polish	11
	J2ME Polish Installation Guide	11
	License Selection	11
	WTK Directory Selection	12
	Component Selection	13
	The Installation	14

External Tools	15
J2ME Polish Sample Applications	15
Testing the Sample Applications	16
Troubleshooting Sample Application Errors	17
Summary	17

CHAPTER 4 Integrating J2ME Polish into IDEs

Recognizing Common Integration Issues	19
Using Eclipse	20
Integrating Eclipse with Ant	20
Troubleshooting Integration Issues	22
Installing the J2ME Polish Plug-Ins	22
Using NetBeans	23
Using JBuilder	24
Using IntelliJ	24
Summary	26

PART 2 ■ ■ ■ Using J2ME Polish

CHAPTER 5 Getting to Know J2ME Polish

J2ME Polish from 500 Miles Above	29
Managing the Application Life Cycle with J2ME Polish	30
Designing the Architecture	31
Implementing the Application	32
Building Your Application	32
Testing the Application	32
Optimizing Your Application	33
Deploying Your Application	33
Updating Your Application	36
Summary	38

CHAPTER 6 The Device Database

Understanding the XML Format	39
Defining Devices	40
Defining Vendors	44
Defining Groups	44
Defining Libraries	45

Describing Known Issues	47
Defining Complementing Capabilities and Features	48
Using the Device Database	49
Selecting Your Target Devices	50
Selecting Resources for Your Target Devices	50
Optimizing for Your Target Devices	51
Changing and Extending the Device Database	52
Summary	53
CHAPTER 7 Building Applications	55
Taking an Ant Crash Course	55
Creating a “Hello, J2ME Polish World” Application	58
Introducing the Build Phases	64
Selecting the Target Devices	65
Preprocessing	66
Compilation	66
Obfuscation	67
Preverification	68
Packaging	68
Invoking Emulators	68
Packaging Your Application	68
Resource Assembling	68
Managing JAD and Manifest Attributes	74
Signing MIDlets	77
Using Third-Party Packagers	78
Building for Multiple Devices	80
Selecting Devices	80
Minimizing the Number of Target Devices	83
Building Localized Applications	84
The <localization> Element and Localized	
Resource Assembling	84
Managing Translations	85
Coping with Dates and Currencies	88
Avoiding Common Mistakes	89
Localizing the J2ME Polish GUI	90
Integrating Third-Party APIs	92
Integrating Source Code Third-Party APIs	92
Integrating Binary Third-Party APIs	93
Integrating Device APIs	94

Obfuscating Applications	94
Using the Default Package	95
Combining Several Obfuscators	95
Debugging Applications	96
Using Conditions	96
Using J2ME Polish As a Compiler	98
Summary	100
CHAPTER 8 Preprocessing	101
Why Preprocessing?	101
Preprocessing Directives	104
Branching Your Code	105
Defining Temporary Symbols and Variables	109
Including Values of Variables in Your Code	110
Using Several Variable Values Individually	111
Including External Code	112
Analyzing the Preprocessing Phase	112
Hiding Statements	112
Logging	113
Setting CSS Styles	113
Nesting Directives	114
Managing Variables and Symbols	114
Using Standard Preprocessing Symbols and Variables	114
Setting Symbols and Variables	115
Transforming Variables with Property Functions	117
Preprocessing to the Rescue!	118
Using Optional and Device-Specific Libraries	118
Changing the Class Inheritance	119
Configuring Applications	119
Using Hard-Coded Values	120
Circumventing Known Issues	121
Summary	121
CHAPTER 9 The Logging Framework	123
Logging Messages	123
Adding Debug Code for Specific Log Levels	125
Controlling the Logging	125
Viewing the Log on Real Devices	128
Forwarding Log Messages	130
Summary	131

CHAPTER 10	Using the Utilities	133
	Utility Classes	133
	The ArrayList Class	134
	The TextUtil Class	135
	The BitMapFont Class	136
	Other Utility Classes	138
	Stand-Alone Utilities	138
	The Binary Editor	139
	The Font Editor	142
	The SysInfo MIDlet	143
	Summary	143
CHAPTER 11	Game Programming with J2ME Polish	145
	Using the Game Engine	145
	Optimizing the Game Engine	146
	Running Your Game in Full-Screen Mode	147
	Using a Back Buffer in the TiledLayer	148
	Splitting an Image into Single Tiles	149
	Defining the Grid Type of a TiledLayer	149
	Using the Game Engine for MIDP 2.0 Devices	150
	Working Around the Limitations of the Game Engine	150
	Porting an MIDP 2.0 Game to the MIDP 1.0 Platform	152
	Porting Low-Level Graphics Operations	152
	Porting Sound Playback	155
	Controlling Vibration and the Display Light	156
	Summary	157
CHAPTER 12	Working with the GUI	159
	Introducing Interface Concepts	160
	Controlling the GUI	161
	Activating the GUI	161
	Configuring the J2ME Polish GUI	162
	Programming the GUI	170
	Using Correct import Statements	170
	Setting Styles	171
	Using Dynamic and Predefined Styles	173
	Porting MIDP 2.0 Applications to MIDP 1.0 Platforms	173
	Programming Specific Items and Screens	174

Designing the GUI	181
Designing for Specific Devices and Device Groups	182
Using Dynamic, Static, and Predefined Styles	183
Extending Styles	187
Reviewing CSS Syntax	187
Common Design Attributes	189
Designing Screens	199
Designing Items	215
Using Animations	228
Summary	229

■ CHAPTER 13 Extending J2ME Polish

Extending the Build Tools	231
Understanding the Extension Mechanism	231
Creating Your Own Preprocessor	239
Setting the Compiler	243
Using a Postcompiler	244
Integrating Your Own Obfuscator	244
Integrating a Preverifier	245
Copying and Transforming Resources	245
Using Different Packagers	246
Integrating Finalizers	247
Integrating Emulators	247
Adding Property Functions	248
Extending the J2ME Polish GUI	248
Writing Your Own Custom Items	248
Loading Images Dynamically	261
Creating Your Own Background	265
Adding a Custom Border	269
Extending the Logging Framework	269
Summary	270

PART 3 ■ ■ ■ Programming in the Real World

■ CHAPTER 14 Overview of the Wireless Market

Introducing Device Differences	273
Hardware	273
Profiles and Configurations	274

Optional Packages	276
The JTWI Specification and Mobile Service Architecture	278
Supported Formats	279
Device Modifications	279
Device Issues	280
The Emulator Trap	280
Examining the Current Market	280
Telecom Market	280
J2ME Market	282
Summary	282
CHAPTER 15 Dancing Around Device Limitations	283
Identifying Vendor Characteristics	283
Nokia	283
Motorola	287
Samsung	288
Siemens	289
LG Electronics	290
Sony Ericsson	290
RIM BlackBerry	291
Other Vendors	292
Identifying Carriers	292
Identifying Platforms	292
MIDP Platforms	293
DoJa Platforms	294
WIPI Platforms	294
Writing Portable Code	294
Using the Lowest Common Denominator	295
Using Dynamic Code	295
Using Preprocessing	298
Using Different Source Files	300
Solving Common Problems	303
Using the Appropriate Resources	303
Circumventing Known Issues	304
Implementing the User Interface	308
Networking	310
Playing Sounds	313
Using Floating-Point Arithmetic	314
Using Trusted MIDlets	317
Identifying Devices	318

Getting Help	321
Honoring Netiquette	321
Exploring J2ME Polish Forums	322
Exploring Vendor Forums	322
Exploring General J2ME Forums and Mailing Lists	322
Summary	323

CHAPTER 16 **Optimizing Applications** 325

Optimization Overview	325
Improving Performance	326
Measuring Performance	326
Performance Tuning	329
Improving Perceived Performance	338
Reducing Memory Consumption	341
Measuring Memory Consumption	341
Improving the Memory Footprint	342
Decreasing the Size of JAR Files	344
Improving the Class Model	344
Handling Resources	349
Obfuscating and Packaging Your Application	350
Summary	350

PART 4 ■ ■ ■ **Appendix**

APPENDIX	353
JAD and Manifest Attributes	353
MIDP 1.0 Attributes	353
MIDP 2.0 Attributes	354
Vendor-Specific Attributes	355
Runtime Properties	356
System Properties	356
Bluetooth Properties	358
3D Properties	359
Permissions for Signed MIDlets	360

The J2ME Polish Ant Settings	363
<info> Section	363
Device Requirements Section	365
Build Section	366
Emulator Section	388
Standard Preprocessing Variables and Symbols	389
Device-Specific Symbols	389
Device-Specific Variables	390
Configuration Variables	392
Symbols and Variables for Reading the Settings	397
Preprocessing Directives	398
Property Functions	399
The J2ME Polish GUI	400
Backgrounds	400
Borders	412
J2ME Polish License	417
Abbreviations Glossary	417
INDEX	419

Foreword

So, you are a developer, you write applications for mobile devices, and you've been asked by a network operator to get your application working on their list of preferred devices, otherwise they won't deal with you. That all sounds familiar and, initially, it all sounds pretty easy—we are developing applications in Java, and as we all know, Java is platform independent and we just “write once, run anywhere.”

OK, now that you've picked yourself up off the floor and stopped laughing after that statement, you've probably realized the big deal about writing mobile applications is that there are so many little differences between devices and the Java implementations on those devices that it's a mammoth task to support many devices.

How do we get around these “inconsistencies”?

How do we avoid having device-specific source code for everything we write?

These are among the *really* important questions for mobile application developers. When I started writing applications for mobile devices, we were still learning about these inconsistencies and still finding things that made us want to tear our hair out.

I would like to suggest that your first port of call should always be the API specifications. You see, that's why they are there: to show you exactly what *should* happen and how you *should* handle things—but as you know, if you sit a thousand monkeys in front of typewriters, you certainly don't end up with the works of Shakespeare. Well, the same applies here. If you give all the device manufacturers the specifications for the application management software, you certainly can't expect them to work in exactly the same way. (Now, I'm not saying it's all their fault—I mean, it's not always exactly the same hardware, and miracles don't happen.)

So, how *do* we get around these inconsistencies? Well, that's where this book and J2ME Polish come in. I recommend going straight to the “Programming in the Real World” section and having a quick read through it. Then you should have a better appreciation for the problems at hand, and you'll realize that you need some help to control the mass of workarounds that you're likely to need. Of course, that help comes in the form of J2ME Polish, and this book explains how to use it.

As I mentioned earlier, when I started writing mobile device applications, we were still figuring out the problems, and to a certain extent we still are. I used to have my own custom Ant scripts for individual builds of an application for individual devices. I just looked at one of the device benchmark sites and it already has results for over 400 devices! I certainly don't want to have 400 versions of source code for one application. I first came across J2ME Polish when I was looking for a better way to implement my build system, and since then I've found it to be an extremely useful addition to my entire development process. Tools like this are what will keep us developing applications in a reasonable amount of time for multiple devices, which keeps the network operators and publishers happy—and you want to keep them happy, because this mobile applications industry is one of the fastest-growing industries in the world.

So if you want to be successful, keep your knowledge up to date, keep your eye on the ball, and keep enjoying the ride. J2ME Polish keeps developers happy and able to make everything work.

Welcome to your world—make it a good one!

Kirk Bateman
Managing Director/Lead Developer
Synaptic Technologies Limited (UK)

About the Author



■ **ROBERT VIRKUS** is the architect and lead programmer for the open source project J2ME Polish. He is an internationally recognized J2ME expert and is a member of Mobile Solutions Group, Bremen, Germany.

After studying law and computer science in Bremen, Germany, and Sheffield, England, Robert began working in the mobile industry in 1999. He followed WAP and J2ME from their very beginnings and developed large-scale mobile betting applications.

In 2003, he founded Enough Software, the company behind J2ME Polish.

In his spare time, Robert enjoys the company of his girlfriend, Jinny, and his dog, Benny. Other spare-time favorites are going to concerts of soul, ska, and punk-rock bands, and playing around with old computers like Atari 400, Commodore 8296, and MCS Alpha 1.

About the Technical Reviewer



■ **THOMAS KRAFT** is a managing director of Synyx, a company located in Karlsruhe, Germany. Synyx specializes in developing business solutions based on open source frameworks. Thomas has many years of experience developing Java applications, with a focus on business solutions and J2ME applications. He is also an expert in the open source content management system OpenCms. Thomas has written several articles about J2ME developing, mostly GUI related. Thomas resides in the beautiful city of Karlsruhe and can be reached via e-mail at *kraft@synyx.de*.

Acknowledgments

Writing this book was hard work, but also a lot of fun. Thanks to everyone who made it possible, especially Jinny Verdonck, Thomas Kraft, Ricky Nkrumah, Kirk Bateman, and the whole Apress crew. A big thanks goes also to the community that continuously extends and improves J2ME Polish!

Introduction

This book introduces J2ME Polish, a collection of open source tools for creating “polished” wireless Java applications. J2ME Polish is best known for its build tools and its user interface—but more about that later. In this book, you will learn how to use J2ME Polish to your advantage. You will also learn about the challenges you will encounter in the real world of wireless Java programming and how J2ME Polish can help you to master these problems, by circumventing device bugs, integrating the best matching resources, and more.

The first part of this book helps you to install and integrate J2ME Polish in your development system. The second part of the book deals with the various tools included in J2ME Polish, including the following:

- Device database (Chapter 6)
- Build tools (Chapter 7)
- Preprocessor (Chapter 8)
- Logging framework (Chapter 9)
- Utilities (Chapter 10)
- Game engine (Chapter 11)
- User interface (Chapter 12)

You can also extend J2ME Polish in almost every aspect, which I discuss in Chapter 13. In the third and last part of the book, you will learn about differences between J2ME devices, known issues, and typical challenges of J2ME programming in the real world.

In this book, I assume that you are already familiar with J2ME development, so you should know what a MIDlet is, what the Mobile Media API is used for, and so forth. This book will not teach J2ME programming; instead, it focuses on how you can get the most out of your programming.

Please feel free to get in touch with me at robert@enough.de.

PART 1



Getting Ready

Creating mobile Java applications is great fun. Use this part for learning how to install J2ME Polish and other necessary or useful tools for J2ME programming. If you already have J2ME Polish up and running, I suggest that you skim through this part, so you don't miss out on the additional tips and tricks.



Quick Setup Guide

In this chapter:

- Download J2ME Polish (<http://www.j2mepolish.org>).
- Install J2ME Polish (double-click the downloaded file or call `java -jar j2mepolish-[version].jar`).
- Check out the sample applications (call `ant` or `ant test j2mepolish` in the *samples/menu* or *samples/roadrunner* directory), if you have Ant and the Wireless Toolkit (WTK) installed.

About two years ago, I was working on yet another mobile application and experienced the agony of J2ME development: the application worked fine on the emulators, but not in the crude real world. Then it worked on one device, but not another. After a while, my coworkers and I managed to get it running on all our target devices (five different handsets), but we had to split up the application into a different branch for each device we targeted. Then we needed to incorporate any changes into each branch. What a headache! We used way too much time coding device adjustments—more time than we devoted to the actual application itself. Being good programmers, we managed it in the end, of course. Then we presented our finished project. Although the application itself was regarded as good, the design was utterly rejected. So, we had to redo the code, once again in every branch.

This scenario probably sounds familiar to you. And, as a programmer, you probably wondered, as I did, if it really had to be that way. Why should you create yet another application branch just for incorporating device-specific adjustments? Why should you do these adjustments in the first place? And last, but not least, why should you, as a programmer, be forced to design the user interface of the application yourself? These are the types of problems that J2ME Polish was designed to address.

In this chapter, you'll get a quick start with J2ME Polish. Here, I assume that you have set up Ant and the Wireless Toolkit already. Please refer to the following chapters for a detailed step-by-step guide.

Installing J2ME Polish

The installation of J2ME Polish is done in three distinct phases:

1. Install the Java Software Development Kit (SDK) 1.4 or higher, the Wireless Toolkit (WTK), and a Java integrated development environment (IDE) or Ant.
2. Install J2ME Polish.
3. Integrate J2ME Polish into your favorite IDE.

You can download J2ME Polish from <http://www.j2mepolish.org> and start the installation by either double-clicking the downloaded `.jar` file or by calling

```
java -jar j2mepolish-[version].jar
```

from the command line. (Sanity note: Please substitute the `[version]` with the actual version number of J2ME Polish.) You should now be presented with the screen similar to Figure 1-1.

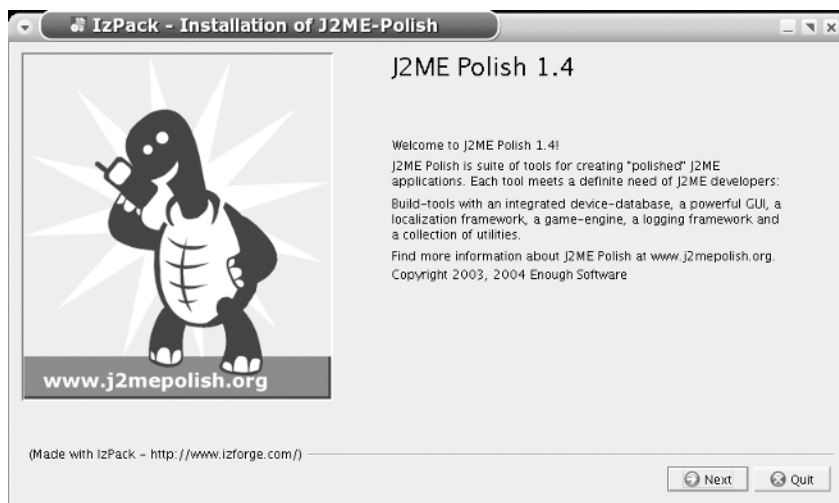


Figure 1-1. Installing J2ME Polish

Launching the Sample Applications

After you have installed J2ME Polish to any directory, which I refer to as `#{polish.home}`, you can check out the sample applications, which can be found in the `samples` directory. Change (`cd`) to one of the directories, such as `samples/menu`, and call Ant from the command line:

```
ant test j2mepolish
```

The default WTK emulator should pop up when J2ME Polish has finished processing the application, as shown in Figure 1-2.



Figure 1-2. The sample menu application in the WTK emulator

If you see any error messages, most likely your Ant setup or your path to the WTK is not correct. See the “Troubleshooting Sample Application Errors” section in Chapter 3 for help.

You can also start J2ME Polish from within any Java IDE. You need to mount the sample project, right-click the *build.xml* file within it, and select Run Ant, Execute, or a similar menu command to launch J2ME Polish. If you want to start the emulator, make sure that you have selected the test target first, followed by the `j2mepolish` target.

Exploring the Sample Applications

If you wonder how the sample applications are made, you can take a peek at the *build.xml* file in either the *samples/menu* or *samples/roadrunner* directory. This standard Ant file controls the build process and uses the `<j2mepolish>` task for creating the application.

If you take a look at the code itself in the *src* directory of the Menu application, you will find a very simple application that uses a `javax.microedition.lcdui.List` for displaying a menu. Open the *resources/polish.css* file to find out how this application was designed. This file contains the design information in a human-readable format. To gain your first experience with J2ME Polish, change the `fontColor` in the `colors` section on the top from `rgb(30, 85, 86)` to red, and then restart J2ME Polish by calling `ant test j2mepolish!`

Congratulations, you are now ready to rumble!

Summary

This chapter explained the necessary steps for installing and using J2ME Polish in a very condensed manner. The following chapters describe how to install the other tools you need to run J2ME Polish, detail the J2ME Polish installation steps, recommend some additional tools that you might find helpful, and show how you can tightly integrate J2ME Polish into your favorite IDE.



Installing the Prerequisites

In this chapter:

- Install the Java SDK (<http://java.sun.com/j2se>).
- Install the WTK (<http://java.sun.com/products/j2mewtoolkit>, or the Mac OS X version from <http://mpowers.net/midp-osx>).
- Obtain an IDE (such as Eclipse, <http://www.eclipse.org>), if you don't already have one.
- Install Ant (<http://ant.apache.org>).
- Obtain device emulators (from vendors, such as <http://forum.nokia.com>, <http://motocoder.com>, and <http://developer.samsungmobile.com>).

J2ME Polish relies on several open-source and free tools. You need to have the Java 2 Software Development Kit (SDK), the Wireless Toolkit (WTK), and Ant—either stand-alone or as part of your integrated development environment (IDE). You should also install device emulators from several vendors for testing your applications. Even though these are not strictly needed for developing great applications, they make your programming life a good deal easier. This chapter covers the installation of these tools.

The Java 2 SDK

J2ME Polish is a solution for creating wireless Java applications. So, it's only natural that you need Java itself as well. Java comes in three editions for catering to different needs:

- The Java 2 Standard Edition (J2SE) runs on your desktop computer and is needed by J2ME Polish for generating the actual mobile applications.
- The Java 2 Micro Edition (J2ME) runs on mobile phones, but also on television set-top boxes, personal digital assistants (PDAs), and embedded devices.
- The Java 2 Enterprise Edition (J2EE) runs on servers and is used for powering web applications or delivering mobile applications over the air.

If you do not have J2SE installed, please download the latest 1.4.x version now from <http://java.sun.com/j2se> and install it. You can also use Java 5.0 (1.5.x), but many mobile emulators require J2SE 1.4, so I recommend that you to stick to the 1.4 branch for now.

■ **Tip** See *Beginning J2ME: From Novice to Professional, Third Edition*, by Jonathan Knudsen (Apress, 2005) for a great introduction to J2ME development.

The Wireless Toolkit

The Wireless Toolkit (WTK) provides not only a generic emulator of a Java-enabled mobile phone, but also a preverification tool, which you need to make a J2ME application ready for deployment.

The installation is straightforward once you have downloaded it from <http://java.sun.com/products/j2mewtoolkit>. Just install it into the default directory, and you're ready to go.

WTK Versions

Several versions of the WTK are available, notably the old 1.0.4 version for Mobile Information Device Profile (MIDP) 1.0 phones and the latest version from the 2.x branch. Usually, you should go with the latest version, but you can also use the 1.0.4 version if you prefer it. The main difference is that you need to compile the code to Java 1.1 when you use the older WTK, whereas Java 1.2 is used by default when WTK 2.x is available. In theory, this shouldn't make a difference, but sometimes you might encounter really mind-boggling bugs that seem to be linked to overloading problems. In such cases, the `javac` target can make a difference, so just keep in mind that you can change it when you use WTK 2.x, but not when you use WTK 1.0.4.

WTK for Mac OS X

The standard WTK is not available for Mac OS X, but you can use a port that is provided by mpowers LLC at <http://mpowers.net/midp-osx>. To do so, you need to have X11 installed as well, which you can obtain at <http://www.apple.com/macosx/x11>.

The WTK itself is delivered as a disc image, which is mounted automatically when you double-click it. Just install it by dragging the mounted disc image onto the desktop first, and then into the *Applications* folder. If you don't have administrator's rights, you can move the disc image into the *Applications* folder within your home folder (create that folder if necessary).

IDE

In case you don't use an IDE yet, I recommend the Eclipse IDE, which is available for free from <http://www.eclipse.org>. Eclipse provides a modular and powerful environment based on plug-ins. J2ME Polish also brings along some Eclipse plug-ins that help you to write preprocessing code, manage emulators, and so on. Chapter 3 discusses the possible integration in more detail. Refer to <http://eclipse-tutorial.dev.java.net/> to learn more about developing Java programs with Eclipse.

Other popular IDEs include NetBeans (<http://www.netbeans.org/>) and JBuilder (<http://www.borland.com/jbuilder/>).

Ant

Ant forms the well-established standard for building any Java applications. You can use Ant from the command line or from within any serious IDE, such as Eclipse, NetBeans, JBuilder, or IDEA.

If you want to use Ant from the command line, you should download the binary distribution from <http://ant.apache.org> and extract it. You then need to adjust your PATH environment variable, so that the ant command from the *bin* directory can be found.

If you have installed Ant into *C:\tools\ant*, enter the following command on your Windows command line (or your shell script):

```
SET PATH=%PATH%;C:\tools\ant\bin
```

You can change the PATH variable permanently in the System Settings of Windows (Start ► Settings ► Control Center ► System ► Advanced ► Environment Variables).

Also set the JAVA_HOME environment variable, which needs to point to the installation directory of the Java 2 SDK:

```
SET JAVA_HOME=C:\j2sdk1.4.2_06
```

Under Unix/Linux/Mac OS X, use the export command instead of the SET command:

```
export PATH=$PATH:/home/user/tools/ant/bin
export JAVA_HOME=/opt/java
```

You can set any environment variable automatically by editing the *.bashrc* script, found in your home folder.

Now you should be able to test your Ant setup by querying the installed version:

```
> ant -version
Apache Ant version 1.6.2 compiled on September 11 2004
```

Note Please refer to the “Ant Crash Course” section in Chapter 7 for a guide to getting started with Ant. It’s not that difficult!

Vendor-Specific Emulators

Emulators let you check out the look and feel of an application, and sometimes even help you find bugs. Some emulators are little more than the standard WTK with another skin, but some do reproduce the actual behavior of the device quite well.

Caution Beware of the “But it works in the emulator” trap! Never rely on an emulator. Test as early and as often as possible on the real device.

You can usually download emulators directly from the vendors' web sites, such as listed in Table 2-1. Some vendors provide additional services, such as discussion forums and the like. Most sites require you to register before you can download any resources. Sometimes, carriers and operating system developers provide emulators too. Most emulators are available for Windows only; some are available for Linux as well. For Mac OS X, you can use the mpowerplayer SDK (<http://mpowerplayer.com>).

Table 2-1. *Emulator Vendors*

Vendor	URL	Remarks
Nokia	http://forum.nokia.com	Look for Java tools and SDKs for getting several emulators. Most common are the Series 60 and the Series 40 developer kits.
Motorola	http://motocoder.com	The Motorola SDK contains several emulators for most J2ME-enabled devices. Go to Tools ► SDK to download the SDK.
Samsung	http://developer.samsungmobile.com	The Samsung site supports only Microsoft Internet Explorer 6.0 and above. KDE's Konqueror also works, but Mozilla-based browsers cannot be used. Check out the Resources section for downloading the SDKs.
Sony Ericsson	http://developer.sonyericsson.com	You can download the SDK from Docs & Tools ► Java.
Siemens	http://communication-market.siemens.de/portal/main.aspx?pid=1	Download the toolkit by choosing Resources ► Tools.
Symbian	http://www.symbian.com/developer	Download the emulators from the SDKs section.

Summary

In this chapter, we looked at the various basic tools that you need in order to use J2ME Polish. In the next chapter, you will learn how to install and set up J2ME Polish itself, so that you can start developing professional mobile applications!



Installing J2ME Polish

In this chapter:

- Get the J2ME Polish installer (<http://www.j2mepolish.org>).
- Invoke the installer (by calling `java -jar j2mepolish-[version].jar`).
- Use the provided sample applications.

Thanks to the graphical installer, the setup of J2ME Polish is relatively painless. This chapter describes the installation of J2ME Polish, as well as some third-party tools that you can use with J2ME Polish.

J2ME Polish Installation Guide

You can download the J2ME Polish installer from <http://www.j2mepolish.org>. Just select Download from the main page.

Tip While you're at <http://www.j2mepolish.org>, you can also subscribe to the mailing lists by selecting Discussion, then Mailing Lists. The `polish-users` list provides help and general discussions about J2ME Polish. The `polish-announce` list keeps you updated about new releases.

You can invoke the installer either by double-clicking the downloaded file or by calling `java -jar j2mepolish-[version].jar` from the command line. You need to substitute the `[version]` part with the real version number, such as `j2mepolish-1.4.1.jar`. Now the installer should start.

The installer is a simple wizard that needs your input for some steps. It presents three critical screens: license selection, WTK directory selection, and component selection.

License Selection

The license selection screen, shown in Figure 3-1, asks you to decide which license you want to use. The license will be included in the `build.xml` files, which control the actual build process. So, you can change the license at any time by modifying these files.