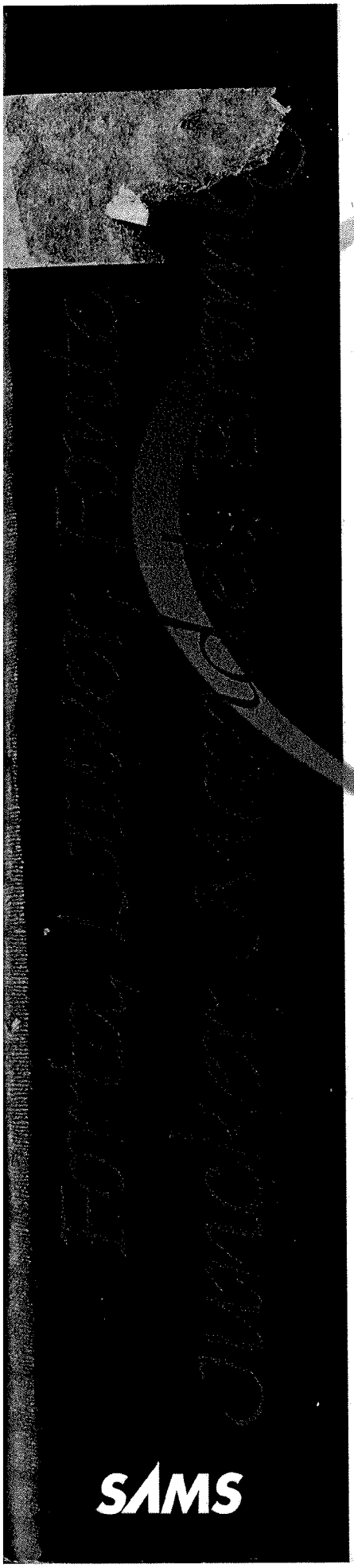


Ben Forta with Keith Lauver,
Paul Fonte, Robert M. Juncker,
Ronan Mandel, and Dylan Bromby

WAP Development with WML and WMLScript

THE AUTHORITATIVE SOLUTION

Master the concepts

WAP 1.2, WML (Wireless Markup Language), WMLScript, Servers and Gateways, Devices and Device Emulators, and more

Build the applications

Create a currency convertor, an online user directory, a personal scheduler, and a phone-based e-commerce application using WML, WMLScript, and a variety of back-end technologies (including ASP, ColdFusion, and JSP)

SAMS

Foreword

The Internet is changing the way we shop, entertain ourselves, and communicate. As the Internet evolves, multiple distinct media are emerging—the PC, the TV, and now the mobile phone. Phone.com was instrumental in this evolution by originating the concept of the browser-enabled phone and co-founding the WAP Forum with Ericsson, Nokia, and Motorola in 1997. We continue this tradition of innovation with our dedicated support of the developer community.

WAP presents a major new market opportunity for the Internet community. The number of mobile phones is forecast to exceed 1.4 Billion by 2003. The characteristics of the mobile phone—its portability, "always on" nature, and its location-sensitivity—present a unique application opportunity.

WAP is more than a protocol or a markup language; it is a new "mobile-centric" Internet application platform. As such, there has been a lack of comprehensive developer-oriented material to guide Internet developers through the process of becoming great mobile-centric application designers—not only the basics, but also a true real-world guide to creating great applications.

Whether you are an Internet or intranet developer, *WAP Development with WML and WMLScript* will serve as an invaluable tool in your exploration of the mobile Internet. This book was written by some of the leading experts in WAP tools and application development, with input from our own experts here at Phone.com. Its real-world approach and hands-on format will greatly benefit anyone interested in developing for this exciting mobile Internet marketplace.

Ben Linder
Vice President, Phone.com
`developer.phone.com`

WAP Development with WML and WMLScript

Ben Forta

with

Keith D. Lauver

Paul Fonte

Robert M. Juncker

Amy O'Leary

Ronan Mandel

Dylan Bromby

SAMS

A Division of Macmillan USA
201 West 103rd St., Indianapolis, Indiana, 46290 USA

WAP Development with WML and WMLScript

International Standard Book Number: 0-672-31946-2

Library of Congress Catalog Card Number: 00-104240

Printed in the United States of America

First Printing: September 2000

03 02 01 00 4 3 2 1

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The authors and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the CD or programs accompanying it.

Associate Publisher
Michael Stephens

Acquisitions Editor
Angela Kozlowski

Development Editors
Gus A. Miklos
Mark Renfrow

Managing Editor
Matt Purcell

Project Editor
Christina Smith

Copy Editor
Michael Dietsch

Indexer
Greg Pearson

Proofreader
Candice Hightower

Technical Editors
Andi Hindle
Aubrey Harden

Team Coordinator
Pamalee Nelson

Media Developer
J.G. Moore

Interior Designer
Gary Adair

Cover Designer
Aren Howell

Graphic Art Conversion Technician
Oliver Jackson

Production
Ayanna Lacey
Stacey DeRome-Richwine

Contents at a Glance

Table of Contents

PART II Creating WAP Applications

About the Authors

Lead Author

Ben Forta is Allaire Corporation's Product Evangelist for the ColdFusion product line. He has over fifteen years of experience in the computer industry in product development, support, training, and product marketing. Ben is the author of the best-selling *ColdFusion Web Application Construction Kit* (now in its third edition), and its sequel *Advanced ColdFusion 4 Development* (both published by Que), as well as books on Windows 2000 development, JSP, Allaire HomeSite, and Allaire Spectra. A member of WAP Forum, and frequent lecturer and columnist on Internet technologies, Ben is finding himself spending more and more time on WAP and wireless application development. Born in London, England, and educated in London, New York, and Los Angeles, Ben now lives in Oak Park, Michigan with his wife Marcy and their five children. Ben welcomes your email at `ben@forta.com`, and invites you to visit his Web site at `http://www.forta.com`.

Contributing Authors

Keith D. Lauver began fifteen years ago at the company that is now known as Gearworks.com. A recipient of the prestigious Cook Scholarship, Keith attended St. Paul's School and later Carleton College, where he majored in Economics. During his time in high school, Keith wrote many custom database applications, and while at college, he developed the company's first commercial database application product. These endeavors would become the foundation for Gearworks.com's focus on data-driven business-to-business solutions. Since 1995, Gearworks.com has grown steadily into a full-service provider of Web and wireless solutions for national and multinational corporations. In 1999 as CEO of Gearworks.com, he refined the company's offerings to focus on mobile application development, leveraging the new emerging devices and standards enabled in part by WAP. Keith's experience includes system architecture, strategic and technical consulting, project management, and development engineering.

Paul Fonte is a Principal Software Engineer with Media Station Inc., located in Ann Arbor, Michigan. He has been architecting, designing, and implementing commercial software systems for over 10 years. Paul received engineering degrees from both Purdue University and the University of Michigan. He built his first wireless Web application back in 1997 using HDML 2.0 for OnTime. With Media Station, he now contributes to the SelectPlay product—a broadband home Internet application that delivers CD-ROMs to home Windows desktops on demand. Paul welcomes your email correspondence at `paul@fonte.com`.

Robert M. Juncker's (Gearworks.com) history with technology dates back to 1990 when he first captured the title, "State Programming Champion of New Jersey." Shortly after, Rob conducted advanced application development for wireless alarm systems. During his time at Carleton College where he majored in computer science, Rob entered the field of speech recognition, consulting with companies like IBM and Lernout & Hauspie. At ViA Inc., Rob was the Software Director for Speech and Language Technologies, where he built a software development warehouse and implemented the first voice-to-voice language translator designed for a truly mobile operation. With over 10 years of computing experience and a deep knowledge of wireless products, including wireless LAN development and PDA application development, Rob has had unique exposure to new technologies building on C/C++, PL/SQL, Delphi, Visual Basic, Codewarrior, HTML, ASP, DCOM, ActiveX, OLE, and ColdFusion. At Gearworks.com, Rob is VP, Technology leading the technical teams in developing innovative wireless solutions that leverage WAP and integrate with existing platforms such as Allaire Spectra and ColdFusion.

Ronan Mandel has been the lead developer support engineer at Phone.com for the past two years. Prior to that, he worked in the JavaSoftware Division of Sun Microsystems from 1995 until 1999. He has been working with internet technologies since 1990 and holds a degree in Geophysics from the University of California at Santa Cruz.

Dylan Bromby is Director of Wireless Product Development for Go2 Systems, Inc. in Irvine, California. Prior to joining Go2, Dylan worked in the advanced technology group of Autobytel.com where he created new consumer Web applications and assisted the exploration of new business models and market opportunities with partners such as Intel Corporation. In addition to working many hours at Go2, Dylan has been a speaker on topics such as e-commerce, Internet start-ups, and WAP on behalf of institutions such as the University of Southern California Marshall School of Business and University of California, Irvine Graduate School of Management. You can visit Go2 at `http://www.go2online.com` on the Internet, HDML and WML-capable mobile phones, or RIM pagers. You are welcome to contact Dylan at his permanent email address at `dylan@bromby.com`.

Dedication

To Nancy, Johnny, Danny, and Mark—the proof that heaven is where the heart is.

—Paul Fonte

To Diana, for all the love through the years.

—Ron

Acknowledgments

First of all I'd like to thank my co-authors Keith Lauver, Paul Fonte, Dylan Bromby, Rob Juncker, Amy O'Leary, and Ron Mandel, for their outstanding contributions. And an additional special thank you to Amy for all that extra last minute work. Thanks to Andi Hindle and Aub Harden for a superb technical editing job. Thanks to Neil Cormia and Kathy Simpson of Phone.com for lots of support and help. Thanks to all the developers and organizations who gave us permission to include their software on the accompanying CD-ROM, thereby increasing the value of this book for all. And finally, thanks once again to everyone at Macmillan, especially Gus Miklos for a superb development job, and J. G. Moore for tirelessly scouring the electronic universe for the software to be included. A very special thank you to my Acquisitions Editor, Angela Kozlowski, who makes it close to impossible to work with other AE's by constantly setting standards that no one else can ever hope to attain.

—Ben Forta

Special thanks to Rocky DeVries, Rob Davis, Jared Parish, Sarah Shomler, Jeb Sawyer, Dave Morehouse, Mike Bollinger, Jeremy Raadt, Andrew Kass, Dave Anderson, Jim Larsen, and Pete Kobs at Allaire. Extra special thanks to everybody at the Contented Cow in Northfield, MN.

—Gearworks.com

Keith Lauver

Robert M. Juncker

First, I would like to thank my family—Nancy, Johnny, Danny, and Mark—for helping me write, each in her or his own special way. I must also thank my great friends at Media Station; I'm privileged to be part of such a great team. Next, I would like to thank Anik Ganguly who started me writing wireless applications with a wild idea one summer weekend. Of course, I need to thank Ben Forta for inviting me into another great writing project. Finally, I would like to thank all of my other co-authors, Angela Kozlowski, Gus Miklos, Michael Dietsch, and all the other fabulous folks at Sams.

—Paul Fonte

I'd like to recognize and thank the thousands of dedicated people around the world who take time out of their personal and professional lives to participate in the WAP Forum and further the collective WAP efforts on so many fronts. Thanks to Ben for the opportunity to get involved with this project. And to Ben and Angela both, I thank you for your patience. I would also like to thank the incredible people at Go2. We all came from such diverse backgrounds and expertise to make something great. Thank you to my parents for always encouraging and challenging me to work hard and put people above everything else. Thank you to my mother and father-in-law who have been great sources of encouragement and love and taken care of my wife when work has taken me away from home. And the biggest thank you of all, I owe to my beautiful wife Ann. I've learned from you, grown with you, and been through easy and challenging times with you. Thank you for your sacrifices and for encouraging me to become who I am.

—Dylan Bromby

I would like to thank my co-authors for helping me through this effort. At Phone.com my deepest gratitude goes out to several folks, (including but not limited to) Paul Smethers, Jolie Bories, Don Schuerholz, Darren Sloan, Jacky Mann, Kathy Simpson, Alistair France, and Ben Linder. Most of all I would like to thank Diana, for all of her love and encouragement.

—Ronan Mandel

Tell Us What You Think!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

As an Associate Publisher for Sams Publishing, I welcome your comments. You can fax, email, or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books stronger.

Please note that I cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail I receive, I might not be able to reply to every message.

When you write, please be sure to include this book's title and author as well as your name and phone or fax number. I will carefully review your comments and share them with the authors and editors who worked on the book.

Fax: 317-581-4770

Email: networking_sams@macmillanusa.com

Mail: Michael Stephens
Associate Publisher
Sams Publishing
201 West 103rd Street
Indianapolis, IN 46290 USA

Introduction

Why We Wrote This Book

Every once in a while a technology comes along that simply captures the imagination. The desktop computer was definitely one of these, as were the introductions of HTTP, the World Wide Web, and the recent explosive growth in wireless communication. And although each of these is exciting by itself, when the three are merged the possibilities are nothing short of mind-boggling.

Wireless data communication is not a new idea. For years now we've been hearing that wireless communication has finally come of age, only to be disappointed over and over again. And yet this time we believe what we are hearing. This time wireless computing is becoming a reality and in ways no one could have imagined.

WAP is the Wireless Application Protocol, a communications protocol (based on HTTP) designed specifically for wireless communication and managed by the WAP Forum. WAP is the transport used to communicate between devices (phones initially, but other devices eventually) and servers. WAP is very technical, but the reason that WAP will succeed where all others have failed is anything but so. WAP will succeed because it is being supported by almost every major hardware, software, device, data carrier, and telecom vendor. And with that kind of muscle behind a common goal, anything is possible.

WAP is exciting, and that excitement is shared by developers, the press, the financial community, and end users alike. But thus far obtaining all the information needed to leverage this new and exciting technology has been rather difficult. A shortage of documentation, unclear guidelines, inadequate real-world advice, and hard-to-find tools have all contributed to complicating the getting-started process. And that's where this book fits in.

Who Should Use This Book

This book was written for developers who want to leverage what promises to be one of the most important protocols and standards ever developed.

If you are an HTML developer and want to learn how to port your sites to WAP, this book is for you. If you are a Web application developer (writing in ASP, ColdFusion, Java, JSP, Perl, or other server-side language) and want to learn how to generate content for wireless devices, this book is also for you. And if you just want to learn more about WAP and related technologies, this book is for you too.

Written by developers for developers, and with input from WAP Forum members (including Phone.com and Allaire), this book teaches you everything you need to know to take advantage

of WAP and all it has to offer. From WAP fundamentals to the details and nuances of the WML language, from using scripting with WMLScript to securing applications, from using alerts to writing for both HTML and WML, from fine-tuning graphics to best practices and user interface guidelines, you'll find more useful content in this volume than anywhere else. And all in an informative and highly code-centric style that makes learning a breeze.

How to Use This Book

This book is designed to be used in two ways. For starters, work through the book systematically, beginning with the very first chapter and continuing from there. More-experienced developers will find that the extensive indexes and cross references make this book an invaluable reference tool.

This book is divided into five primary parts:

Part 1: Getting Started

This part covers the basics of WAP development and includes three chapters:

Chapter 1, "Understanding WAP," introduces the basis of WAP and the components that make it work.

Chapter 2, "Introducing WML," introduces the Wireless Markup Language and explains some of the basics do's and don'ts of this language.

This is continued in Chapter 3, "Writing for WAP in WML," which walks you through creating your first WML card using basic language elements.

Part II: Creating WAP Applications

This part covers important application development fundamentals and includes four chapters:

Chapter 4, "Card Navigation," teaches the use of actions and key assignments.

Chapter 5, "Managing Output," covers card setup and layout, and text formatting functions.

Images must be used very carefully in WAP, and Chapter 6, "Using Images," explains how to use images and local icons in detail.

Chapter 7, "Working with User Input," teaches how to collect data from users, including working with variables containing user input.

Part III: Advanced WAP Development

With the basics covered, this part teaches advanced WAP topics, and includes seven chapters:

Chapter 8, "Email Integration," teaches the use of provider mail services to both send and receive email from devices.

WMLScript is a scripting language that can be used to extend your WML code, and Chapter 9, "Using WMLScript," introduces this language.

Chapter 10, "Using Timers," teaches the use of timers for greater program control and automation.

Chapter 11, "Receiving Notifications," covers the differences between push and pull and the use of notifications.

Security is an important part of application development, and Chapter 12, "Securing Applications," teaches application security in detail.

Chapter 13, "Writing for HTML and WML," explains how to port code from HTML to WML and how to write code for both platforms at once.

This theme is continued in Chapter 14, "Best Practices," which details some of the hard-learned important do's and don'ts of WAP development.

Part IV: Sample Applications

This next part walks through four complete applications to help you jump-start your own development:

Chapter 15, "Currency Converter," is an example of a client-side application using both WML and WMLScript.

Chapter 16, "User Directory," is a complete database-driven directory application written in Microsoft ASP.

Chapter 17, "Scheduling," is a time-management and scheduling application written in Allaire ColdFusion.

Chapter 18, "E-Commerce," is a sample electronic commerce application written in JSP (with mirror copies in ASP and ColdFusion on the CD-ROM).

Part V: Appendixes

This final part contains seven useful appendixes:

Appendix A, "Wireless Markup Language Reference," is a complete alphabetical list of all WML tags with syntax, examples, and usage notes for each.

Appendix B, "WMLScript Library Reference," is a complete list of available WMLScript libraries and their functions, and even includes coverage of Nokia and Phone.com extension libraries.

Appendix C, "Using Device Emulators," covers the use of the three major emulators (from Ericsson, Nokia, and Phone.com).

Appendix D, "Writing WML in Popular Development Tools," explains how popular development environments can be used to write WML.

Appendix E, "Writing WML in Popular Development Languages," teaches how to write WAP applications in Allaire ColdFusion, Microsoft ASP, Perl, Java, JSP, and PHP.

Appendix F, "Local Icons," is a complete alphabetical reference of the local icons supported by some devices.

Appendix G, "CD-ROM Contents," lists the programs and files included on the accompanying CD-ROM.

So, turn the page, start reading, and welcome to the wonderful world of WAP and wireless application development.

Special Conventions Used in the Book

The people at Sams and Macmillan USA have spent many years developing and publishing computer books designed for ease of use and containing the most up-to-date information available. With that experience, we've learned what features help you the most. Look for these features throughout this book to make it easier to read and understand and to help enhance your learning experience.

- Screen messages, code listings, and command samples appear in `monospace type`.
- Uniform Resource Locators (URLs) used to identify pages on the Web and values for HTML attributes also appear in `monospace type`.
- Terms that are defined in the text appear in italics. *Italics* are sometimes used for emphasis, too.
- In code lines, placeholders for variables are indicated by using *`italic monospace type`*.
- Console or keyboard input appears in **bold** type.

Supplemental Information

Throughout the book, you will see cross references to other chapters, sections, and Web sites to help you broaden your knowledge of the topic. Source code listings that appear on the CD-ROM that accompanies this book will be noted. In addition, you will see other helpful notes and supplemental information as follows:

NOTE

Notes present useful or interesting information that isn't necessarily essential to the current discussion, but might augment your understanding with background material or advice relating to the topic.

TIP

Tips give you advice on quick or overlooked procedures, including shortcuts.

CAUTION

Cautions warn you about potential problems a procedure might cause, unexpected results, or mistakes that could prove costly.

Code Continuation Character

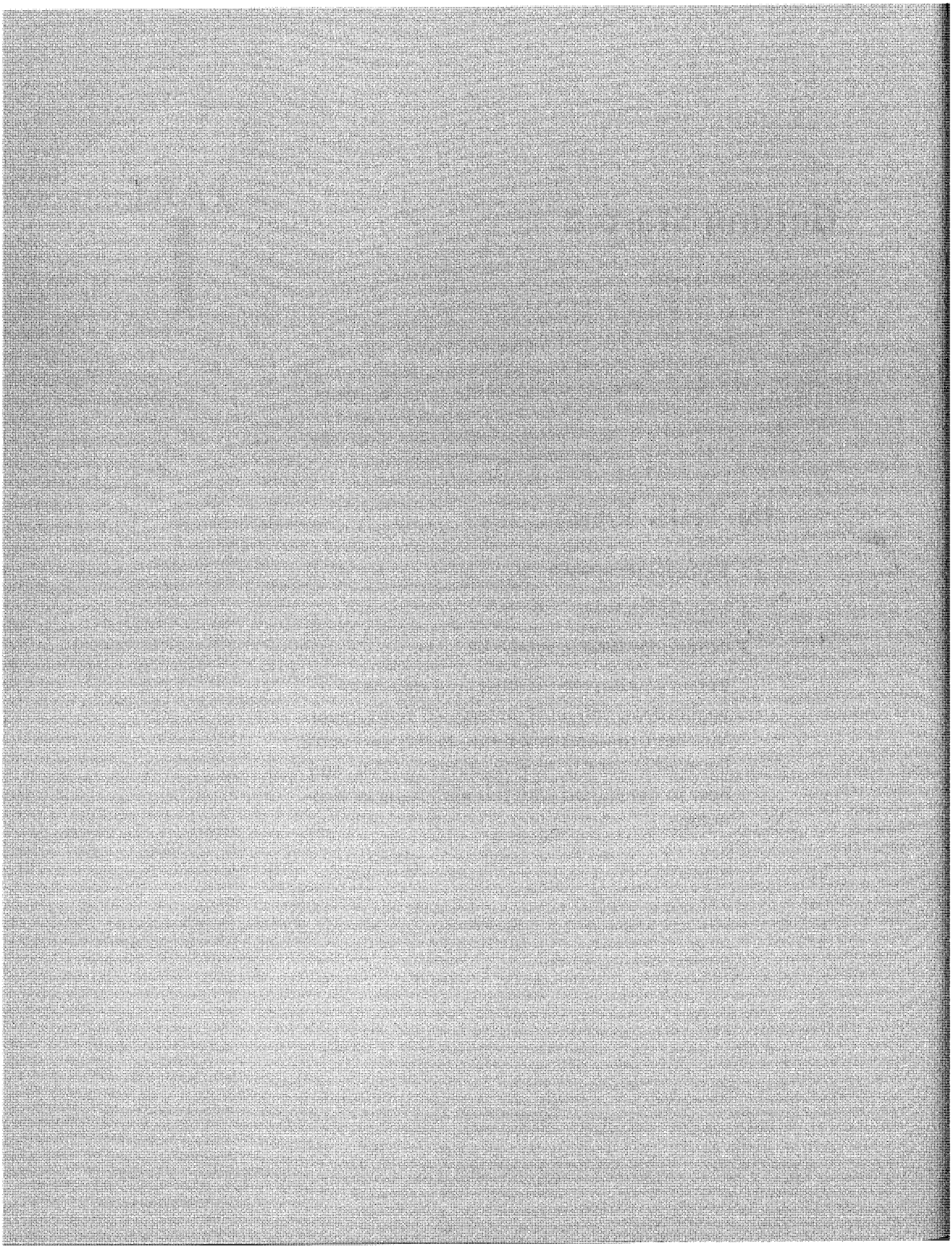
There are many source code listings in this book. Because of space limitations, some of the code lines had to be wrapped (continued on the next line). In actual use, however, you would enter this code as one extended line without a line break. To indicate such lines, a special code continuation character (➥) has been used. When you see this character at the beginning of a line, it means that line should be added, as one line, to the line that precedes it.

PART I

Getting Started

IN THIS PART

Before you can start writing your own WAP applications, you must understand some important basic concepts and terms. In this part you'll learn what WAP is, how it relates to WML, and how to get started with this exciting new technology.



Understanding WAP

CHAPTER

1

IN THIS CHAPTER

WAP stands for *Wireless Application Protocol*, the de facto standard for wireless computing managed by a consortium of vendors called the WAP Forum. WAP does for wireless devices what HTTP does for Web browsers—it allows them to become clients in an Internet-based client/server world.

So what exactly is WAP? WAP is a protocol, a data transport mechanism. In many ways it is similar to HTTP (the protocol that the Web uses for data transport), and WAP was also built on top of established standards, such as IP, URLs, and XML. But WAP was designed from the ground up for wireless computing, and was built to accommodate the unique and fundamental limitations of wireless computing:

- Devices with limited processing power and memory
- Limited battery life and power
- Small displays
- Limited data input and user interaction capabilities
- Limited bandwidth and connection speeds
- Frequent unstable (lost or poor) connections

The job of the WAP Forum is to manage the evolving standards, while ensuring the highest degree of interoperability.

The WAP Forum

The WAP Forum was formed in 1997 by Ericsson, Motorola, Nokia, and Phone.com (at the time known as Unwired Planet). Within two years, more than 100 companies had joined the group, which will define the standards for providing Internet content and services to wireless devices. Member organizations now include almost every major software, hardware, and device vendor, along with all major carriers and telecom providers.

The WAP Forum's Web site (`http://www.wapforum.org/`) is the primary resource for WAP related documentation and specifications.

WAP is actually not a single protocol; rather, it is a collection of protocols and standards that make up a complete lightweight protocol stack (see Table 1.1) along with special markup and scripting languages (as you'll see later in this chapter), which together define a complete solution.

TABLE 1.1 WAP Protocol Stack

Abbreviation	*NameDescription*
WAE	Wireless Application Environment Application layer, includes the micro-browser on the device, WML (the Wireless Markup Language), WMLScript (a client-side scripting language), telephony services, and a set of formats for commonly used data (such as images, phone books, and calendars).
WSP	Wireless Session Protocol Session layer, provides HTTP 1.1 functionality, with basic session state management, and a facility for reliable and unreliable data push and pull.
WTP	Wireless Transaction Protocol Transaction layer, provides transport services (one way and two way), and related technologies.
WTLS	Wireless Transport Layer Security Security layer, provides data security and privacy, authentication, as well as protection against denial-of-service attacks.
WDP	Wireless Datagram Protocol General transport layer.

WAP is exciting, but for the most part, unless you are writing WAP servers or creating WAP devices, the WAP protocol itself will not be of much interest to you.

Understanding Servers and Gateways

WAP devices connect to servers to retrieve and send information in much the same way as Web browsers connect to HTTP servers. In fact, WAP devices can connect to both WAP and HTTP servers (a behavior designed to eliminate as many barriers to WAP adoption and acceptance as possible).

If you want to serve WAP content you can install a WAP server. This is a piece of software, much like an HTTP server (and indeed, the two can usually run on the same machine). As seen in Figure 1.1, the WAP device makes a request from the WAP server which returns the requested data to the device for processing.

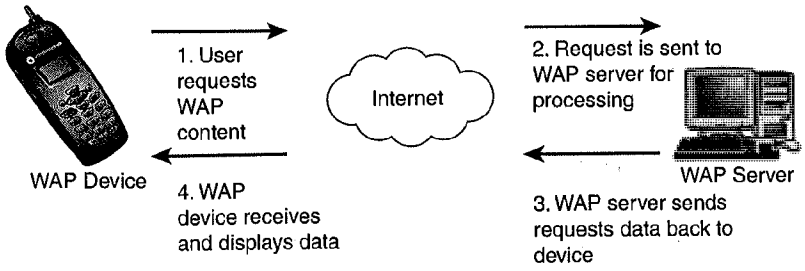


FIGURE 1.1
WAP devices request and receive data from WAP servers.

So if devices talk WAP (and not HTTP) how can they request data from HTTP servers? The answer is that a WAP gateway sits in between the WAP device and HTTP server and acts as an interpreter between them (as seen in Figure 1.2). The WAP gateway handles all data forwarding and filtering or conversion so that what the device gets back is just WAP, not HTTP at all.

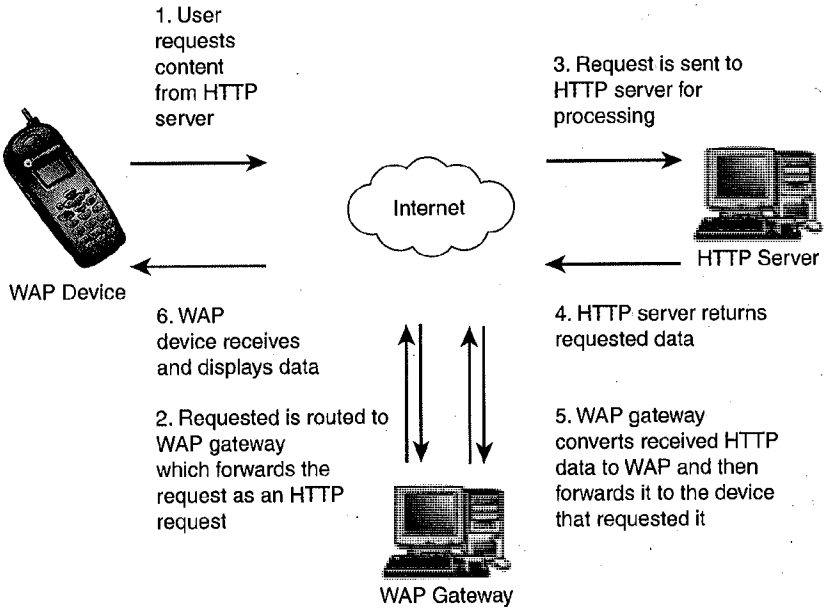


FIGURE 1.2
WAP devices can request and receive data from HTTP servers via WAP gateways.

More often than not, to write WAP applications you'll not need a WAP server or gateway. In fact, you'll probably not need anything more than your current HTTP server. The next chapter, "Introducing WML," covers configuring your Web server so that it serves WAP content.

NOTE

In the same way that HTTP servers can be used to serve WAP content, many of the tools and development environments and languages you use to develop Web applications can also be used to develop WAP applications. This includes Perl, ASP, ColdFusion, Java, and more. Appendix E, "Writing WML in Popular Development Languages," covers the basics of writing for WAP in many major platforms (complete with sample code for each).

WAP Devices

WAP clients are devices—for now phones, but in the future other wireless devices too. These devices all have two common characteristics:

- An integrated browser, called a micro-browser
- A mechanism for user input, which can range from a couple of buttons on simple models to entire banks of buttons with roll bars and touch screens on higher-end (and future) models

As you can imagine, with each device being different, the capabilities and features on each differs too. WAP is designed to be very device independent; the code you write for one device should work on all devices. But there is a big difference between working and working as intended. Different devices implement different features in different ways, and that makes for a very inconsistent development environment. As such, WAP developers must test the code they write on as many devices as possible.

NOTE

HTML developers are often bothered by how little control they have when it comes to generating WAP content. Exact screen placement is impossible, as is guaranteed consistent layout. This is not a flaw in the spec; it is deliberate and by design, for it allows for the widest variety of devices and capabilities. But it also means that developers have to be a little more flexible (and creative) when designing user interfaces. And they have to stop comparing WAP to the Web.

Table 1.2 lists some of the more commonly available devices, and their primary features. Of course, new devices are being developed all the time, so don't read this list as a definitive list of what's available—it's not.

TABLE 1.2 Some Popular WAP Devices

Device	*Notes*
Alcatel OneTouch	Good text formatting and alignment, and supports local icon use. Features the Phone.com UP WAP browser. Available in Europe.
Casio C303CA	Good text formatting and alignment, and supports local icon use. Features the Phone.com UP WAP browser. Available in Japan.
Denso C202DE	Good text formatting and alignment, and supports local icon use. Features the Phone.com UP WAP browser. Available in Japan.
Denso TouchPoint	Good text formatting and alignment, and supports local icon use. Features the Phone.com UP WAP browser. Available in the U.S.
Ericcson R280	Good text formatting and alignment, and supports local icon use. Features the Phone.com UP WAP browser. Available in the U.S.
Ericcson R320s	Features a five-line display, basic organizer functionality, and good text formatting and alignment. Small screen size.
Ericcson R380	Features a wide pressure-sensitive screen under the keypad, and good text formatting and alignment. Also includes a complete organizer based on the Psion 5 running EPOC 32.
Hitachi C201H and C302H	Good text formatting and alignment, and supports local icon use. Features the Phone.com UP WAP browser. Available in Japan.
Mitsubishi T250 and MobileAccess	Quad mode phone. Good text formatting and alignment, and supports local icon use. T9 text-input software. Features the Phone.com UP WAP browser. Available in the U.S.
Motorola i1000 plus	Good text formatting and alignment, and supports local icon use. Features the Phone.com UP WAP browser. Available in the U.S.
Motorola Timeport	Very small form factor. Good text formatting and alignment, and supports local icon use. Features the Phone.com UP WAP browser. Available in U.S.

Device	*Notes*
Motorola Timeport L7389	Very small form factor. Good text formatting and alignment, and supports local icon use. Features the Phone.com UP WAP browser. Available in Europe.
NeoPoint 1000	Large screen with 11-line display. Good text formatting and alignment, and supports local icon use. Features the Phone.com UP WAP browser. Available in the U.S.
Nokia 6185	Good text formatting and alignment, and supports local icon use. Features the Phone.com UP WAP browser. Available in U.S.
Nokia 7110	Features a roll bar for menu selection, predictive text input, and a large high-contrast screen. Limited text formatting and alignment. Uses Nokia's own WAP browser.
Qualcomm QCP-1960 and QCP-2760	Good text formatting and alignment, and supports local icon use. Features the Phone.com UP WAP browser. Available in the U.S.
Samsung Duette and SCH 3500	Good text formatting and alignment, and supports local icon use. Features the Phone.com UP WAP browser. Available in the U.S.
Samsung SGH-800	Good text formatting and alignment, and supports local icon use. Features the Phone.com UP WAP browser. Available in Europe.
Sanyo D301SA and TS01	Good text formatting and alignment, and supports local icon use. Features the Phone.com UP WAP browser. Available in Japan.
Sanyo SCP-4000	Backlit display and large directional control. Good text formatting and alignment, and supports local icon use. Features the Phone.com UP WAP browser. Available in the U.S.
Toshiba C301T and TT01	Good text formatting and alignment, and supports local icon use. Features the Phone.com UP WAP browser. Available in Japan.

Obviously, owning every device out there is extremely difficult (if not impossible, as many devices are only available in specific countries). Testing your code on every single available device is a great idea; it's also a terribly impractical one.

One partial workaround is the use of device emulators (like the one shown in Figure 1.3), software programs that simulate WAP devices on your computer. Some of these even support the use of skins (replaceable screens with different interfaces) so that multiple devices can be tested with a single emulator.

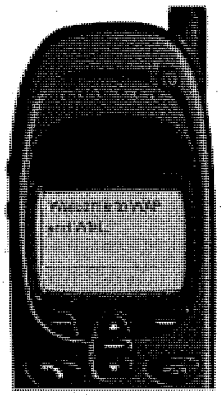


FIGURE 1.3
Device emulators are ideal for testing WML code.

NOTE

See Appendix C, "Using Device Emulators," for a list of emulators and how to use them (as well as where to obtain them).

WAP and WML

WAP is a protocol, just like HTTP (and most developers will spend as much time worrying about WAP as they do HTTP—in other words, not much at all). The truth is, WAP is pretty boring: It's a protocol, it works, and it facilitates data transfer.

Just as Web developers spend most of their time writing HTML, most of the action in WAP development is not at the WAP level; it's at the markup language used to create WAP applications. As WAP devices have special user interface requirements and restrictions, the use of HTML is not an option. For example,

- WAP devices don't have pointing devices, so mouse-like interfaces are impossible.
- WAP devices do not have full keyboards; they usually have simple phone keypads along with some extra keys.

- WAP device screens are small. They can't support frames, complex tables, large graphics, and sophisticated color and font control.
- WAP devices have no real multimedia, sound, or video support.

Knowing all these limitations and restrictions, could you even consider writing applications in HTML? Obviously wireless devices have special needs, and so a special markup language was created for WAP—WML, the Wireless Markup Language.

As an XML-based language loosely derived from HTML, WML is tag based and uses familiar tag pairs and attributes for all language features. WML uses many of HTML's tags (`<p>`, `<table>`, and `<em>` to name a few), but WML is not as extensive as HTML, and many of the tags that are included behave differently and have different attributes.

As WML is tag based and based on HTML, it is easy to learn and use (although it is far stricter in syntax and usage than is HTML). WML features support for

- Cards (WAP pages) and decks of cards (sets of WAP pages in a single file) containing formatted text (covered in Chapter 3, "Writing for WAP in WML," and Chapter 4, "Card Navigation")
- Images (in special formats, as will be discussed in Chapter 6, "Using Images")
- Data entry (discussed in Chapter 7, "Working with User Input")

▶ WML is covered extensively in this book, starting with Chapter 2, "Introducing WML," and Chapter 3, "Writing for WAP in WML."

WML is complemented by a client-side scripting language called WMLScript (just like HTML is complemented by client-side JavaScript). WMLScript is far simpler in scope than JavaScript, but it does provide basic programmability that you can use to perform basic text and data manipulation.

▶ WMLScript is covered in Chapter 9, "Using WMLScript."

To help understand all the pieces in the WAP puzzle and what they mean, Table 1.3 compares WAP technology and features with their familiar Web counterparts.

TABLE 1.3 Comparing WAP to the Web

Feature	*Web*	*WAP*
Transport	HTTP	WAP
Markup	HTML	WML
Scripting	JavaScript	WMLScript

NOTE

For some reason, perhaps because it is easier to pronounce than WML, people refer to all wireless development as "WAP" (when they are usually talking about WML). When you hear WAP, remember that WAP is a transport, and what most developers concern themselves with is WML (and WMLScript).

Summary

WAP is the de facto standard for wireless computing. Made up of data transport and supporting protocols, along with specialized markup and scripting languages, WAP is a complete solution for wireless development. In the next chapter you'll learn about WML, and what you need to start writing code.

Introducing WML

CHAPTER

2

IN THIS CHAPTER

This chapter introduces Wireless Markup Language (WML) by discussing its origins, functionality, and similarity to other common programming languages you may already know. This chapter also helps you get started developing your own WML code by showing you how to configure your Web server for WML content and how to view that content with several common devices and device emulators. Finally, we'll show you how to determine what kinds of applications are relevant to the special capabilities of WML, while covering basic development concepts, such as syntax and WML's particular card and deck structure.

What Is WML?

Wireless Markup Language (WML) is a markup language used for describing the structure of documents to be delivered to wireless devices. WML is to wireless browsers as HTML is to a browser on a desktop computer. WML was created to address the display, bandwidth, and memory limitations of mobile and wireless devices, such as cellular phones and wireless handheld computers. Because it was designed to run on a variety of devices, WML assumes very little about the device running the application and provides much less control over output formats than you might be used to with HTML.

WML's Origins

In the early 1990s, Unwired Planet created *HDML (Handheld Device Markup Language)* to serve as the development standard for wireless applications. By June 1997, Unwired Planet had changed its name to Phone.com and, along with Nokia, Motorola, and Ericsson, launched the WAP Forum—a nonprofit organization dedicated to the development and proliferation of a single standard protocol for wireless applications. Using Phone.com's HDML as the basis for its own standard markup language, the forum created and distributed WML—a language different from, but in many respects similar to, HDML. The WAP Forum and detailed specifications for WML can be found on the Web at `http://www.wapforum.org/`.

NOTE

WML is similar to HTML, but Web developers will find WML less forgiving and more strict when it comes to syntax.

Although WML will certainly look familiar to Web developers accustomed to HTML, the two languages are really more like cousins than brothers. SGML (Standardized Generalized Markup Language) can really be thought of as the father of both HTML and XML (Extensible Markup Language). HTML is designed to handle a lot of objects, pictures, and other multimedia, which makes it too bulky for the bandwidth limitations of current mobile devices; therefore, HTML was rejected as the basis for WML, which needs a simple architecture that

structures data to aid the parsing of a document. That need, and the desire for a language that would survive the demands and fluctuations of turbulent standardization discussions, was the reason that WML was based on XML. By using XML as a base, WML was designed to be a lightweight protocol that would meet bandwidth limitations of existing mobile devices.

WML's Functionality

WML supports six key areas:

- *Text presentation and layout*—Although specific devices and WML browsers vary in their output of WML code (very much like output differences between Netscape Navigator and Internet Explorer), line breaks, text formatting, and alignment are all supported by WML.
- *Images*—Although WAP-compliant devices are not required to support images, WML supports the Wireless Bitmap (WBMP) image format and image alignment on the screen. Wireless Bitmap is a graphic format created by the WAP Forum which is optimized for mobile devices.

  ▶ For more on using images in WML, see Chapter 6, "Using Images."

- *User input*—WML supports choice lists, multilevel choice lists, text entry, and task controls.
- *Card and deck organization*—User interactions are divided into cards, and navigation occurs between cards. Decks are related sets of cards which constitute a single file, like a single HTML file. In HTML, viewing one page is akin to viewing one card in WML. However, instead of each HTML page constituting one HTML file, multiple WML cards constitute one WML deck, which is then saved as a single file.
- *Navigation*—WAP supports the standard Internet URL naming scheme and anchored links, allowing navigation between cards in a deck, between decks, or between other resources on the network.
- *State and context management*—To maximize network resources, WAP allows for variables to be passed between WML files. Instead of sending a complete string, variables can be sent and substituted at runtime. The user agent can cache both variables and WML files, minimizing cache hits and server requests. It is also possible to pass variables between different cards in the same deck; this is an important way to minimize network usage.

Similarity to HTML and XML

If you are already familiar with HTML, SGML, or XML, you should recognize WML syntax, which is based on XML but is more formally defined than a general XML application.

If you are accustomed to developing in HTML, several key differences in WML will take some getting used to. Most fundamentally, the target screen for your application is severely limited and yet variable. Current WAP-compliant devices have display screens that range from sub-VGA graphics to text-only displays with four lines and a maximum of 32 characters per line. Second, the multiplicity of devices running these applications makes it hard for developers to know exactly how their application will appear to any given user. WML developers are not dealing simply with two browsers, as HTML developers typically do, but with dozens and dozens of devices—each with a unique screen and interface. WML also introduces new kinds of functionality on the browser side. WML already provides for several complex actions that would traditionally be coded into your application with a scripting language.

Finally, WML is less forgiving than HTML in several ways: Most obviously, it does not compensate for incorrectly nested tags or uppercase usage.

Getting Started with WML

To begin writing WML, all you really need is a simple text editor such as Microsoft Notepad. However, a developer toolkit will get you up to speed more quickly and provide better familiarity with the WAP environment. While these toolkits don't help you actually write WML, they do provide tools to view the WML you've written in a specific WAP-browser compatible environment. Nokia, Motorola, Ericsson, and Phone.com all offer free developer toolkits to test their browser and device functionality, which require a quick registration at each of their Web sites.

NOTE

Most companies require some form of online developer registration before you can download their SDKs.

Developer toolkits do not provide shortcuts for writing WML; however, some WML editors are beginning to appear on the market—either as extensions to existing HTML editors (such as Allaire's HomeSite), or as independent WML-editing products, some of which are available on the Internet as freeware. Using these products are not required but may be helpful to you.

▶ For more information on developer toolkits and their uses, see Appendix D, "Writing WML in Popular Development Tools."

When you register with Phone.com's Developer Program, you can download the UP.SDK from its Web site. The UP.SDK is a free kit that allows Web developers to create HDML and WML applications. The SDK includes the UP.Simulator that simulates the performance of

UP.Browser-enabled devices. When enrolled at the Nokia WAP Developer Forum, you can download the Nokia WAP Developer Toolkit, which comes with references on general WAP issues, as well as the WML and WMLScript command syntax. It also calls for JRE (Java Runtime Environment) v.1.2.2 or higher, which is available for free download as well.

The Nokia WAP SDK 2 includes a light WAP server, which enables you to read WML decks off your local drive or any HTTP server. Also included is an emulator for the Nokia 7110 phone. The emulator reproduces not only the visual appearance of the 7110, but also the bugs—useful for developers wanting to avoid them.

To download the Ericsson Service Development Kit, you need to register at Ericsson's Developer's Zone. Ericsson's WapIDE SDK consists of three main components: a WAP browser, an application designer, and a server toolset. Motorola's platform is called Mobile Internet eXchange, or *MIX* for short. Its Mobile Application Development Kit is intended to provide an integrated development environment for both voice and data applications.

Using one of these SDKs is typically the quickest way to familiarize yourself with the development challenges of the WAP environment and to develop successful wireless applications. However, toolkits can be intended for creating applications for WAP devices in general, not any one mobile device in particular. The applications you create with developer kit might not look anything like the output on an actual WML browser in a mobile phone. To be absolutely certain that the outputs fit with your intentions, you need to test the applications on an actual WAP device, such as a mobile phone or an emulator (to be discussed later in this chapter).

Configuring Your Web Server

The easiest way to serve WML contents is to use an HTTP server. Unless your WAP site is listed on a portal maintained by a mobile network provider, users will need to type the URL of your site on their device manually—a complicated task with only a phone's buttons to interface with. Although `www` is the de facto standard hostname for HTTP servers, `wap` seems to be emerging as a comparable standard for servers containing WAP applications.

TIP

Most WML pages use the naming convention of `wap.sitenamehere.com`.

However, simply throwing your WAP applications up on a standard HTTP server will not allow users to view content on their phones. The server must "tell" the WML browser that it is about to receive a WML page and *not* an HTML page or some other kind of content. This communication is done using MIME extensions that must be added to your server for WML browsers to correctly read your content.

MIME Types for WML, WMLS, and HDML

MIME stands for Multipurpose Internet Mail Extension, which is a piece of header information that was originally used in email to allow for proper formatting of non-ASCII messages over the Internet. There are many predefined MIME types in common use, such as JPG graphics files and HTML files. In addition to email programs, Web browsers also support a variety of MIME types. This permits the browser to display or output files in formats other than HTML.

MIME types common to Internet servers include

- "text/html" for HTML files
- "image/jpg" for JPG files
- "image/gif" for GIF files

Because these file types are so common, most Web servers are already configured to send the correct MIME types. WAP, however, requires its own MIME types to recognize various file content. Because WAP is still emerging for widespread use, you will most likely have to configure your own server to run WAP applications. By adding these MIME types to your server, different devices will be able to properly interpret and therefore display WAP content.

WAP Version 1.1 requires the five MIME types shown in Table 2.1 to serve WML, WMLScript, and Wireless Bitmap images.

TABLE 2.1 File Types and Corresponding MIME Types

File Extension	*Content Type*	*MIME Type*
wml	WML source code	`text/vnd.wap.wml`
wmls	WMLScript source code	`text/vnd.wap.wmlscript`
wbmp	Wireless Bitmaps	`image/vnd.wap.wbmp`
wmlc	Complied WML	`application/vnd.wap.wmlc`
wmlsc	Complied WMLScript	`application/vnd.wap.wmlscriptc`

Adding the MIME Types to Various Servers

In the following sections, we will detail how to add MIME types to the Apache HTTP Server, the Microsoft IIS Server v4.0, the Microsoft IIS Server v3.0 and below, and Microsoft Personal Web Server v4.0. For any additional servers, refer to your server's documentation.

The Apache HTTP Server

For versions of Apache older that 1.3.4, you should do the following:

1. Edit the `srm.conf` file.
2. Find the `AddType` section and add the following piece to the file:

```
# MIME Types for WAP
AddType text/vnd.wap.wml .wml
AddType text/vnd.wap.wmlscript .wmls
AddType image/vnd.wap.wbmp .wmbp
AddType application/vnd.wap.wmlc .wmlc
AddType application/vnd.wap.wmlscriptc .wmlsc
```

3. Save the file and restart the Apache HTTPd.

Apache recommends that new MIME types be added using the `AddType` directive rather than changing the `mime.types` file, however, it is not a requirement and some administrators may prefer to change the `mime.types` file directly.

If you are using Apache 1.3.4, the contents of the three server configuration files, you can make these changes within a single file named `httpd.conf`.

The Microsoft IIS Server

To add MIME types to the MS IIS Server, follow the steps listed here:

1. On the server console, open the Management console or Internet Service Manager Tool.
2. On the Management console you can define the MIME types to be valid for the entire server or valid only for separate directories.
3. To add a new MIME type to a specific directory, right-click the desired directory and select Properties.
4. Select the `HTTP headers` tab.
5. Click the File Types button near the lower-right corner.
6. Select New Type and the following field values:

   Associated Extension: `.wml`

   Content Type (MIME): `text/vnd.wap.wml`
7. Click the `OK` button.
8. Repeat steps 6 through 7 for each additional MIME type.
9. Reboot your system if instructed to do so.

NOTE

To see all the MIME types currently defined for the server, select the Action drop-down menu from the main console window and select Properties and then File Types. This is also an easier way to add global MIME type mappings.

Microsoft Personal Web Server v4.0

To add MIME types to the MS Personal Web Server v. 4.0, follow the steps listed here:

1. Use the MetaEdit tool in the Microsoft IIS Resource Kit.
2. Launch MetaEdit.
3. Open `/MIMEMAP` under `/LM`.
4. Select MimeMap.
5. Using the pop-up dialog box, add any of the following values to the MimeMap list:

```
wml,text/vnd.wap.wml

wmls,text/vnd.wap.wmlscript

wbmp,image/vnd.wap.wbmp .wmbp

wmlc,application/vnd.wap.wmlc

wmlsc,application/vnd.wap.wmlscriptc

hdml,text/x-hdml
```

6. Click the `OK` button.
7. Reboot your system if instructed to do so.

IIS v3.0 or Below, or Personal Web Server v1.0

To configure the MIME type for HDML and WML files in the Windows Registry, do the following:

1. Run the regedit utility (`REGEDIT.EXE`).
2. Open the registry key:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\
➥InetInfo\Parameters\MimeMap
```

3. Add new String Values with the following names:

```
text/vnd.wap.wml,wml,,5

text/vnd.wap.wmlscript,wmls,,5

image/vnd.wap.wbmp,wmbp,,5

application/vnd.wap.wmlc,wmlc,,5

application/vnd.wap.wmlscriptc,wmlsc,,5

text/x-hdml,hdml,,5
```


NOTE

For all new string values types you add, the string value should be left blank.

4. Reboot your system.

Phones and Emulators

When it comes time to test your applications, you can use either a phone or an emulator that simulates the phone's behavior on your computer.

If your applications are intended for public use, you will want to test your applications on as many WAP devices (or their emulators) as you can. Just as a Web developer should test HTML documents on various browsers and computers, you should test WML documents on as many devices and browsers as possible. Because of the variety of manufacturers, the differences between the output on two WML browsers will be much greater than the differences between the output for two HTML browsers.

Phones

Phone output displays vary depending on their physical capabilities and the browser they are running. As shown in Figures 2.1 through 2.5, many phones have different screens and distinct user interfaces.

FIGURE 2.1
Common WAP phone.

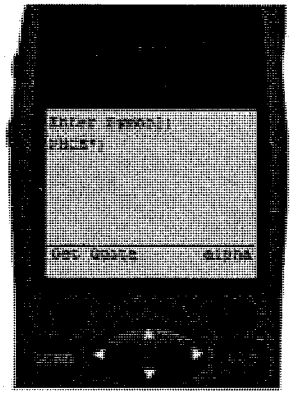


FIGURE 2.2
Generic phone.

FIGURE 2.3
Mitsubishi phone.

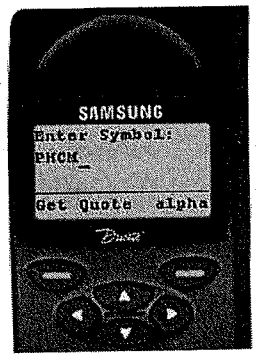


FIGURE 2.4
Samsung phone.

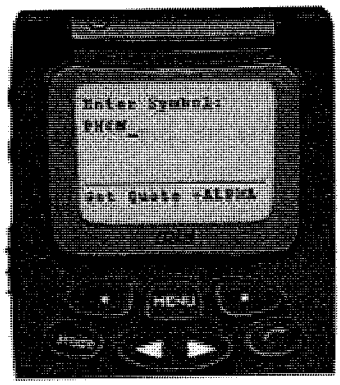


FIGURE 2.5
Motorola phone.

▶ For more about the features and differences between various devices and microbrowsers, see Appendix C, "Using Device Emulators."

Emulators

Emulators are designed to imitate the specific behavior and functionality of mobile devices. Emulators can be used simply to browse WAP sites with your desktop computer and can be especially useful when network coverage is unreliable or if the devices are too expensive for your personal testing use. Some emulators are provided by device manufacturers, and very closely (and in some cases exactly) replicate the behavior of a phone. Other types of emulators that are not provided by device manufacturers can show you what your WML code looks like on your computer, but not in relation to any particular phone or device.

CAUTION

Some "emulators" are really just WML-to-HTML conversion tools and do not replicate the behavior of any particular device. These browsers are better suited for WML surfing than they are for testing real-world applications.

Understanding WML Usage

End users of your WML code have several needs that are unique to wireless application development. To develop truly usable applications in WML, an understanding of how these applications are used in the real world is essential.

To develop a great application, you should keep in mind several fundamental things. First and foremost, a great mobile application will be highly usable. Because of the display and interface limitations on most phones and PDAs, users have a more difficult time determining the context of the application and their options. Clarity and brevity are critical in writing text and titles for WML applications to ensure ease-of-use. Furthermore, your application should require minimal text entry for the user. On many phones with as few as 15 buttons, text entry is highly tedious and anything you can do to minimize that burden for your user is beneficial.

CAUTION

Avoid long URLs for your files. Long URLs increase download time and might cause the application to reach a download limit, thereby preventing the user from accessing a page.

Mobile Users

The Wireless Application Protocol was designed to service needs of mobile users that were not served by previous attempts at mobile computing. These users need specific information that is easily accessible with a multifunctional device that meets most of their needs. When you examine mobile devices that have been on the U.S. market for the last five years (including laptops, rugged computers, or wearable computers), they all present problems to everyday users, consumers, and mobile office workers that are solved by WAP and WAP-compliant devices.

What problems with the old devices make WAP-compliant devices so appealing?

- *Size and bulk*—The older devices are bulky and designed to fulfill a few tasks you would normally do sitting at a desk.
- *Power supply*—A powerful and large device such as a laptop can race through a battery; therefore, heavy battery packs are necessary.
- *Durability*—Because of the size of the device, they are prone to fall and get bumped or damaged in transit.

WAP-compliant devices have been designed to alleviate these issues for users to whom they present a barrier. The devices (most typically phones or PDAs) are small and lightweight, require a reasonable power supply, and are sufficiently rugged to handle the environmental demands mobile usage requires. Because WML uses low bandwidth and power, it is ideally suited to deliver mobile applications to users. With that in mind, how can you design your applications to not only meet but exceed your users' needs?

Imagining Your User

Depending on what kind of application you are building, your users and their needs will vary. The best mobile applications serve users' needs based on *their* priorities—not the developer's. How can you find out what those priorities are? While big companies might be able to conduct focus groups and do research to discover these priorities, individuals developing WAP applications on their own have two courses. One, gain experience as a user and use a phone or emulator to surf existing WAP sites, discovering what works and what doesn't from the user's perspective. Two, use your imagination to create a rounded picture of your users to identify their needs and create a truly relevant mobile application.

For example, instead of developing your coffee-shop locator application for "generic consumer," develop it with several potential users in mind:

> "Mr. Rich Bean"—He commutes to work, has a mobile phone for his business and uses it to check his stock quotes. He's got an expensive corporate phone with all the features. He never turns his phone off, except on weekends.
>
> "Sue Burban"—A mother of two who juggles her family's schedule in their SUV, Sue has an inexpensive phone and has it on only while she's shuttling her kids back and forth in the car. Because she's always on the go, Sue likes to use drive-thru services as much as possible.
>
> "Phil DeMugg"—A college student whose parents gave him a medium-class phone. He's mostly interested in going to the local coffee shop between classes to study and meet up with friends. Phil doesn't have a car and stays near campus most of the time.

By keeping specific user needs in mind, the coffee-shop locator can be built around their needs. Maybe for Sue, you can build in a search or sorting component for coffee shops that have drive-thru windows. Maybe Rich would like to know which coffee shops have special

offers on his route to work. Perhaps Phil would want to know at which campus coffee shop his friends are hanging out or where he can get the cheapest grande mocha.

With more realistic and specific user profiles, you can develop your applications around the kinds of lifestyles and routines your users have, making your applications much more usable.

TIP

Classify the activities end users will use based on need, likely use, and optional or occasional use. Use this classification to prioritize your application's navigational flow.

Relevant Applications

WML answers most of the usability questions of previous mobile-computing solutions, but WAP makes some assumptions to answer them. Small screen sizes solve several problems, but present a new one: limited displays for your application. Relevant WML applications solve this by succinctly providing critical data through superior navigation and sorting. Alternatively, applications that provide lots of content for browsing will prove tedious for the end user and taxing for the phone's bandwidth, rendering the application all but useless.

In another example, WML's limited offline memory presents a challenge to creating a relevant application. For the majority of existing applications, we assume that you will always have access to the digital cellular network. In Europe and parts of Asia, this assumption is entirely valid. In the United States, some suburban areas and most rural areas lack coverage at present and, therefore, constant connectivity cannot be assumed. Applications in the United States, therefore, must be targeted geographically to be relevant and designed to handle instances where the user will be cut off from the network during usage. This may mean that your applications will need to provide a mechanism for reentering the application at the point at which it was cut off.

To address this issue when developing WML applications, it is important to consider the context as well as the usage of the application. For example, an application requiring constant connectivity in Montana would be problematic because of the offline data storage limitations WML presents in poor coverage areas. However, equipping a company's entire sales force with a sales-tracking application and mobile intranet for a company in Minneapolis or New York would be a much better target application as guaranteed data rates, and connectivity will most certainly exist. Although this problem is expected to be alleviated over time as network coverage increases, you must currently work around it when deploying an application for public usage.

CAUTION

Make sure that coverage for wireless data exists in your target area before deploying your application.

What makes an application a good target for a mobile application? There are four basic questions to ask yourself:

- Does the user need real-time access to important data?
- Can the data be presented in an intuitive way onto a device that has screen limitations?
- Can a small amount of data be transmitted to serve the users' needs?
- Is the user going to have excellent cellular data coverage?

Each question seeks to fit the application into an important mobile model.

TIP

WAP sites built strictly for browsing are a poor target for a mobile application. "Brochureware"-style sites—common to the World Wide Web—have much less relevance as WAP sites. Make sure your application has a function that a mobile user might need.

The basis of the Wireless Processing Model in Figure 2.6 is that there is a specific user who needs to access specific information. That information will be manipulated so it can be delivered on a small device with limited interface and small screen size. The key to delivering a relevant mobile application is delivering "golden nuggets" of information. What is "golden" depends entirely on the user and context. If a user is looking for a single fact or statistic in an article, it is always better to streamline the transmission and receipts by delivering only that "golden nugget" because of network bandwidth limitations.

After data is processed, the data is then sent to the user. In this particular model the transmission must also contain device requirements.

Device Functionality

Although it's possible that future mobile devices will be designed with better displays and interfaces, most phones currently on the market have limited interfaces. For example, the NeoPoint 1000 has an extraordinarily robust screen compared to other existing WAP phones. It boasts 11 lines of vertical text with 16 characters across the screen in its minimum-sized font, whereas most others support only four to five lines of vertical text.

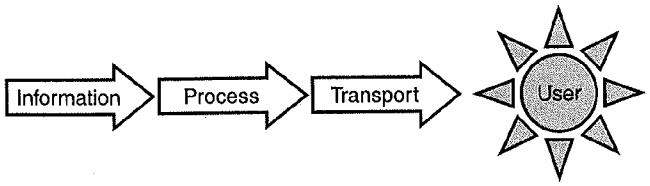


FIGURE 2.6
The Wireless Processing Model.

TIP

Screen sizes can vary widely. Make your assumptions based on the smaller screens, which hold only four to five lines of text, to make sure all users can use your application easily.

Although the NeoPoint has a generous display, it offers a keypad that is still rather awkward to use. For example, to type the letter *l* you would have to push the "5" button on the keypad three times. Additionally, most phones offer only two buttons to navigate menus. One normally functions as a `Select` button, which leaves one button to interact with on menus. The NeoPoint is a good example of the specific limitations of devices and the lack of output control that WML gives you; all other devices on the market have specific limitations and features that must be dealt with.

Limited data entry is one of the reasons your applications must have an easy-to-use and intuitive interface. Data entry is tedious, and for that reason you must try to limit the amount of data entry required in your applications.

CAUTION

Avoid too much text entry. More than the most minimal text entry requirements will frustrate your users.

Any data entry task converted into a choice list or dynamic entry is a better course for your application. For example, Figures 2.7 and 2.8 show two different strategies for answering the question "What is your favorite fruit?"

The style in Figure 2.7 uses manual data entry, whereas Figure 2.8 shows a drop-down menu to solve the problem. In this case, our favorite fruit is an apple. In a test we conducted using manual data entry, it took on average about 15 seconds (and nine button presses) to enter the

needed data. Remember, that test was conducted with individuals highly accustomed to the data-entry format of WAP phones. For new users it will certainly take longer and cause greater frustration. Using the style shown in Figure 2.8, we needed only two button presses, taking about three seconds total, to answer the question.

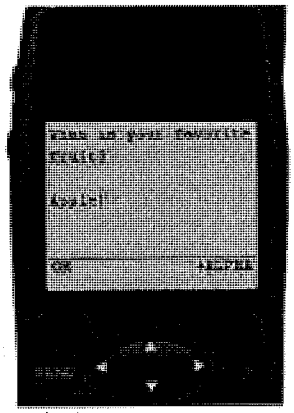


FIGURE 2.7
Data entry styles—manual data entry.

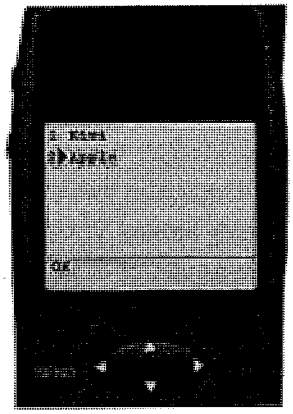


FIGURE 2.8
Data entry styles—dynamic data entry.

CAUTION

When designing WML applications, use check boxes and radio buttons whenever possible and avoid the temptation of asking the user to input raw data. Also, where text fields are necessary, have the application prefill the text fields with default responses.

WML can be extraordinarily powerful when you are writing applications that connect people. Let's examine how WML can help transform the typically frustrating experience of contacting a customer-service call center into an easy and quick experience. How many of us can tell stories of being lost in a touch-tone jungle? Well, that's because 80% of the information we consciously remember is perceived visually. That's just one reason audio phone trees are so frustrating. WML not only has a visual screen but the power to dial your phone for you, allowing for some interesting possibilities to arise. Imagine if you could punch the call center up on your WAP phone using WML and then visually view the menu choices. When you reach the end of the menu choices, your phone would dial the customer service representative and allow any information you entered on your WML screen to instantly appear on the customer service rep's computer screen to better serve you and save everyone's time!

TIP

WML devices have multiple functions. The most powerful applications will use them all!

WAP devices are powerful multifunction devices. The call-center example demonstrates one way these features can be exploited to enhance your applications. The key to developing strong applications is to identify the application's goals and, at the same time, negotiate the device's limitations. Assuming these concepts are well understood, WML is a powerful and easy language that will allow you to easily deploy applications in a mobile environment.

WML Overview

In the next section, we discuss the concepts behind WML and the syntax rules governing WML, and we introduce the most unique aspect of WML—its "card" and "deck" navigational structure.

Concepts

Since WML uses an XML vocabulary, it would be useful to understand some basic principles of XML (Extensible Markup Language), a tag-based system used for defining, validating, and sharing document formats. Although they are very similar, WML differs from XML in the following ways:

- WML's white-space handling rules are not as elaborate as XML's.
- WML relies on *well-formed* expressions.
- WML has a built-in method for handing international characters.

If you are already familiar with the requirements of XML, you might find some portions of the section below repetitive. However, you may want to read through it anyway as the section covers issues specific to WML which are not generically part of XML.

Syntax Rules

WML is like a strict grammar teacher. You will be punished if you do not strictly adhere to the rules set before you. In the next section, we will show you how to be successful in meeting WML's demands to build a working WML application.

Character Sets

WAP, as a protocol, has already found international acceptance. As such, certain international characters are supported, and characters such as tildes (~) and ampersands (&) need to be visible to users.

By default, WML pages use the ISO-10646 character set. In fact, if we were being syntactically perfect in writing our WML code, we would always state this specifically by including in the XML definition at the start of every deck:

```
<?xml version="1.0" encoding="iso-10646"?>
```

The ISO-10646 character set is the computer industry's fancy name for the Unicode character set. This should give you all the basic international characters, but if you need to support a different ISO standard character set, you can adjust this header.

Much like HTML, WML has some reserved characters. For example to print a > in HTML, you would actually escape-code the character and put `&gt` in the HTML code. This holds true for WAP as well. In WML, certain character references are widely supported. These are shown in Table 2.2.

TABLE 2.2 Reserved Characters in WML

Character	*WML Abbreviation*
"	&quot
&	&amp
'	&apos
<	&lt
>	&gt
Space	

Additional characters are also supported through standard escape coding. For example, the word *café* would be written as `caf&#233`, where decimal character 233 in the ISO-10646 character set is the letter é.

For additional information, we recommend you take a look at the Character Entity Reference Chart available online at the World Wide Web Consortium's Web site at `http://www.w3.org/International/O-charset.html`.

White Space

As mentioned earlier, WML differs from SGML quite significantly when it comes to handling white space. In general, if a user were to transmit SGML, all the irrelevant characters would be removed. This is not the case with WML. WML in most incarnations will transmit all the characters in the WML file and then rely on the client device (be it a WAP browser, Web browser, or another application) to display the information in the proper manner, removing irrelevant white spaces; otherwise, line breaks, tab characters, and regular spaces get passed to the application.

TIP

Although white space is transmitted, be sure to use tabs to improve code readability.

Programming Considerations

As we mentioned earlier, WML is syntactically oriented and relies on well-formed code. This means that as developers, we need to be concerned with our spacing of characters when we write in WML. For example, in WML, you must space value pairs with a space, but you cannot use white space between an attribute, the equals sign, and the attributes value.

```
<access
     domain="dsn"
     path="wapdir"
>
```

The previous example is well-formed. The following example, however, would generate errors because of the spaces surrounding the = symbols.

```
<access
     domain = "dsn"
     path = "wapdir"
>
```

Likewise, we cannot run the attributes elements together as below:

```
<access
     domain="dsn"path="wapdir"
>
```

This code would create an error.

WML does make special considerations for the quote characters. Under HTML, the standard evolved ad hoc to correct common errors made by programmers. For example, under Internet Explorer 5.0, the commands

```
<table width="100%">
```

and

```
<table width='100%'>
```

have the same effect. This programming standard has held true for WML as well. In general, it is preferred to use this syntax:

```
<p align="right">
```

But, it is acceptable to use this syntax:

```
<p align='right'>
```

Remember that WML is particular about white space, but at the same time, it is important to adhere to standard coding conventions. For example:

```
<p>
     E-mail $emName<br/>
     Subject: <input type="text" name="subject"/><br/>
     Message: <input type="text" name="message"/>
</p>
```

This code adheres to standard coding conventions of indenting various nested levels. In this case, we see the paragraph's contents are indented further. In WML, it is important that each tag has a match and the nesting order is appropriate. In HTML, we could get by with syntax like this:

```
<B><I>Hello World</B></I>
```

However in WML, strict adherence to nesting levels is required and closing the bold tag before the italics tag would generate an error. Therefore, the only way these commands could work is through the syntax:

```
<b><i>Hello World</b></i>
```

Case Sensitivity

WML is like a grammar test. Any punctuation, any misspellings, and any case-sensitive mistakes will cause the compile to fail. This is particularly true when you consider start and end tags. In HTML, a valid tag match might include <BODY> and </body>. However, in WML, <IMG/> and <img/> would indicate two different start and end tags.

Variables follow the same logic regarding case sensitivity.

```
Variable1
variable1
VARIABLE1
```

All the previous examples are different variables in WML, and each could have a discrete value assigned to it.

Required Prologue

The WAP server and WML compilers need to have a manner by which they can reference the XML specification. To do this, XML requires a prologue that defines the XML version and a pointer to the XML definition or language being used. Most of the examples we refer to for the remainder of this book will use the standard prologue:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">
```

In general, a WAP device will never see this prologue, as the server and compiler are the only two devices that care about the definition of the XML format.

For the sake of brevity, the prologue will be left out of our code examples in this book with the exceptions of the upcoming deck examples.

WML Components

When developing for the Web, each HTML file constitutes one HTML page. Developing in WML is slightly different. Because each *page* or screen is very small, it does not make as much sense for each page to constitute a separate file. WML pages—content viewed on separate screens—are called *cards* and those cards are all placed within a *deck* of related pages which constitute one single file.

Decks

When HTML is transmitted to a Web browser, we consider it to be an Internet document. A deck is the simplest wireless document. It consists of the same basic elements you would expect to see in an HTML document: prologue, WML tags <wml>, header tags <head>, and a few other tags unique to WML. Each deck has a series of cards in it. Therefore, the goal of

WML due to wireless limitations is to keep decks small, and occasionally have a few decks in a single WML application.

Cards

Inside each deck are one or more cards. The cards are the drivers of the application. Each card defines how a particular screen looks, how that screen functions, and what steps are taken when the card is navigated. As such, a card can never be empty and must contain at least one element. The information inside the elements derive either content or navigation instructions.

Summary

This chapter introduced Wireless Markup Language (WML) by discussing its origins, functionality, and similarity to HTML and XML. Next, the chapter discussed how to configure your Web server to serve WML content and how to view that content with several common devices and device emulators. Finally, we showed you how to determine what kinds of applications are relevant to the special capabilities of WML, while covering basic development concepts, such as syntax and the basics of WML's particular card and deck structure.

Writing for HTML and WML

CHAPTER

13

IN THIS CHAPTER

Providing one URL for the users of your application is one thing, but actually being able to deliver appropriate content for a variety of devices from that same URL is another issue all together. The holy grail of Web development is indeed the ability to "write once, display anywhere," and while to some extent this may be a fantasy, some guidelines can help you get closer to that goal.

Just as there are several browsers out there that render HTML in different ways, not all WAP browsers render the same WML content in the same way. This chapter will not tell you how to deliver content to every client out there, but it will lay the framework that will allow you to deliver content in a meaningful way to just about any device.

There are many books available that will provide you with all the guidelines, sample code, and suggestions that you could ever imagine for developing HTML applications. If that's what you're interested in, you should go get another book. The focus of this chapter will not be how to write good or compelling HTML, but instead how you can build your site and/or data structures so that you can readily deliver both HTML and WML from the same data store.

NOTE

It is worth noting that the concepts and ideas discussed here are useful beyond the world of WAP. As new standards for data communication and presentation evolve, so will the number of delivery types and transports you'll need to work with. By paying special attention to creating reusable and portable data upfront you'll be ready for future technologies as they start to appear.

The principle that must be involved is the same as that of CSS, DHTML, DOM, or XML, depending on what you are most familiar with. It boils down to the process of abstracting the presentation layer from the data. There are many HTML browsers out there with varying levels of support for CSS in different formats. It will be some time before you can rely on all Web browsers to understand XML/XSL, so until that happens, you will have to live with the fact that everyone is going to have a different level of capability when they reach your site. How do you serve them all? What do you serve them? Is it all relevant? You must ask yourself these questions to build a successful site.

Why Two Languages?

First of all you must ask yourself, "Why bother writing WML at all? Can't I just use HTML and a translator?" The answer is yes you can, but if you want to build a real-world, compelling application, any translator is not going to get you far enough.

Consider, for example, Amazon.com. Figure 13.1 shows what its Web site looks like in Internet Explorer.

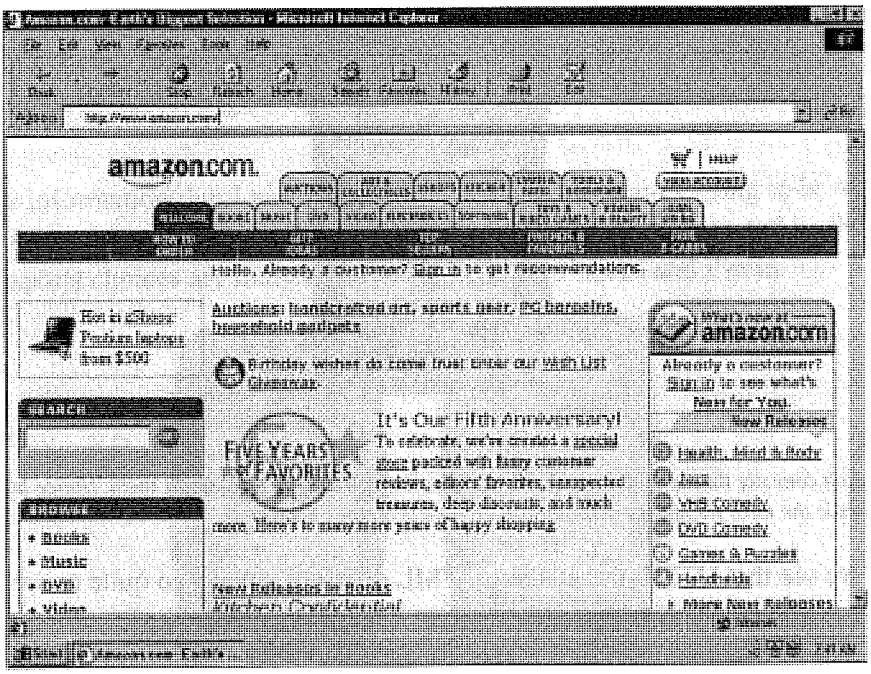


FIGURE 13.1

Amazon.com as viewed through Internet Explorer. There's lots of useful information here that a user can get to quickly and simply.

Figure 13.2 shows what the Amazon.com site looks like when it comes through an HTML/WML Translator. Figure 13.3 shows a version written specifically for WML.

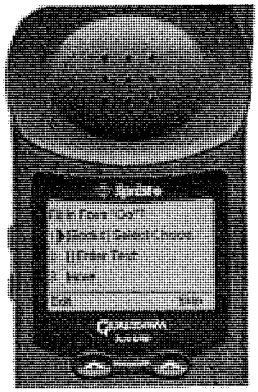


FIGURE 13.2

From an end-user perspective, it is very difficult to understand what you're looking at, let alone how to get to the features of the Amazon.com site.

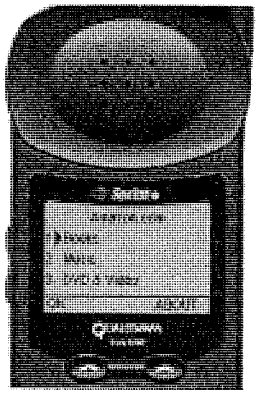


FIGURE 13.3
This is the Amazon.com site that is written explicitly for phones with a WAP browser in them. Notice how the first thing that the user sees is the most frequently used feature of Amazon.com— Books.

Ok, so now we see that HTML translated to WML does not always go quite as well as you might expect. Although it will get you something, the results can be far from optimal. Other sites that do not rely on image maps or that have links defined in an order on the page that is also meaningful may translate better. Also, if a user has experience with your HTML site, he will be better able to navigate through it via translation. Take, for example, Yahoo.com (see Figure 13.4). If I already know that I expect to find a Sports link on the site, and I am persistent, I can find it.

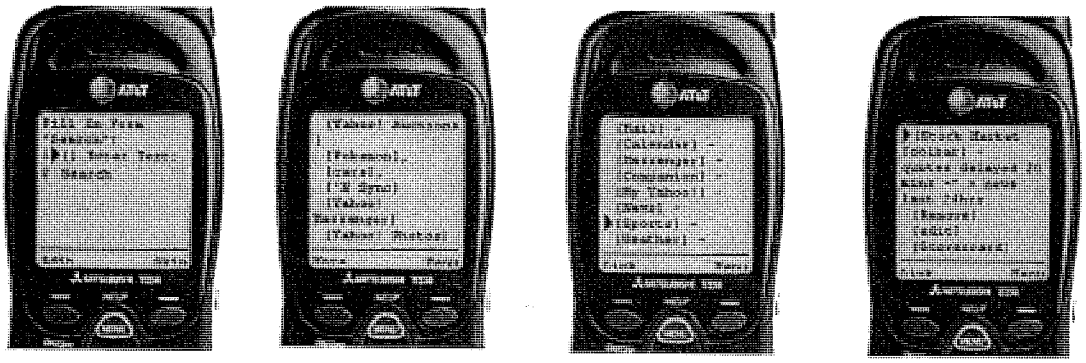


FIGURE 13.4
Navigating through Yahoo.com.

I'm not trying to pick on Yahoo here, almost any site is going to perform this way through an HTML translator. The first thing that the device picked up on in the first phone display is the Search box at the top of Yahoo's page. If I skip this form, I start getting the anchors at the top

of the page as seen in the next display. If I know what the HTML Yahoo site looks like, I can be persistent and choose More to get another page of links, as you can see in the third display, and then choose stock quotes page (last display), and I even left out a few cards for simplicity's sake.

So, I found the information that I was looking for, but how often am I going to be willing to put up with this? In the wireless version of Yahoo, the Sports link is presented on the first page, and my sports choices are immediately available. Two clicks and I'm in.

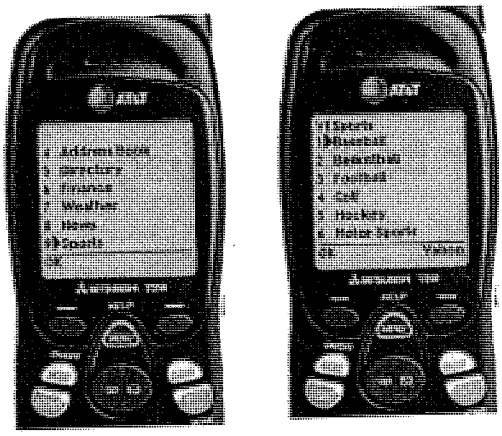


FIGURE 13.5
Since Sports was option 9 on the main Yahoo! page, I had to scroll down to find it. When I select option 9, I am presented with a meaningful menu to choose from.

Clearly, Yahoo! has done some considerable work here to build a powerful wireless site that works as a companion to its HTML site. When I access the content from my phone, I don't get the banner ads, I don't get the extraneous links, and I get direct access to the content that I want to see the most.

The preceding examples were all done through an HTML/WML Translator that was running at a WAP gateway, not at the origin server. It is certainly possible to build a more intelligent translator, and anything that lives at your Web site (rather than at an outside gateway) is going to provide you with a greater level of control. However, this should address the issue of why you should bother writing WML in the first place.

Users who access the Internet from their personal computers by and large are dialing into an ISP and are using somewhere between a 28.8k and a 56k modem. This represents a two- to six-fold increase over the 9600kbps and 14.4kbps speed that HTML browsers had when they became available to the general public in 1994 and 1995. Five years later, we're still using HTTP (1.1 instead of 1.0) and we're still using HTML, GIF, and JPG. When I'm delivering graphical content to a Web browser, I'm still going to use a compressed, optimized format instead of something like a TIFF. The same principle will hold for the wireless environment.

As bandwidth and device capability increase, today's innovations are not going to be thrown away. They will be improved up on and extended.

To this end, the WAP Forum is working very closely with the W3C in the definition of XHTML, also known as HTML5.0. The specification for WML 2.0 will likely be part of the XHTML Specifications. However, remember that we are still talking about the future, and you want an application today. Even after the standards are set and the WAP browsers are built, the product cycle for getting new software into phones is something like 12 to 18 months. So even if the browsers existed today, it would be another year until the phones hit the market. This is one of the reasons why the new WAP phones today support the WAP 1.1 specifications, which were first proposed a year ago; the 1.2 specifications have since been approved.

How to Write for Both Languages

Hopefully, you are convinced that even if you have an existing, sophisticated HTML site, providing a WML interface to your content (rather than relying on translation) is a good idea. You probably would not be reading this book if you didn't already think so anyway. So when you start building the WML side of your application, there are some things to think about.

Phone Considerations

The WAP client is a phone first and foremost, and for your application to be compelling it needs to take this fact into account. You can cause the phone to make a phone call, you can deliver a real-time alert to some phones, and phones have numbered keypads, not keyboards. How you present your data is significant. When you think about delivering content to a phone, think about how the phone is going to be used.

TIP

Employ the code that allows users to generate a call from their phone as often as you can. If you're presenting a phone number, wrap it in `<a href="wtai://wp/mc;`*phone number*`">`*phone number*`</a>` (replacing *phone number* with an actual number) so that the user can call the number by selecting the link.

Consider the following:

- Unlike a PC keyboard with an Enter key and mouse navigation, the phone has one or more soft keys to which actions can be bound.

  An HTML page gives the user the ability to choose from several possible actions with Input/Submit fields, links, and buttons. How should these features be mapped on to a WML card?

- What is the most relevant information on a page that needs to be presented onto the phone?

  Clearly, everything on the HTML page is not going to work on a phone.

- What about frames?

  Which frame should be sent to the phone?

  Is it possible to get back and forth across the frame pages?

- What about banner ads? How would these work?

  Nobody is going to advocate removing banner ads from your HTML pages (this may be one of your current revenue sources), but with limited screen real estate you should think twice before using them.

PC Browser Considerations

Keep It Simple Stupid is one of the guidelines that HTML developers have a strong tendency to avoid. There are plenty of sites out there that are ultra slick and super confusing. The temptation to employ all of the newest/latest technology can be too much to resist at times, and developers have the tendency to overlook the fact that their user base may or may not have the ability to handle this latest technology on their client. I'm not suggesting that you regress to the days of Lynx and build a text-only site, but I do recommend that you consider the usability implications of a clean, easily navigable site. And I can't say it often enough: Just as you would with your HTML code, test your code on as many browsers/platforms/clients as you possibly can. Don't just test it to see whether it breaks, test it for usability. Can uneducated (about your site, not in general) users accomplish simple tasks? Do they get lost in your site? Do they get overwhelmed by your content or presentation?

Are end users going to be driving while they use the browser (I hope not), or are they going to be sitting in a coffee shop? Are they going to be waiting for a train? They're not likely to be at a desk. They are going to use their phones to find specific information quickly, or possibly they are going to want to kill some time and be entertained. Extra information that might enhance your HTML site will likely clutter a WML site.

Exactly what data you present to users on an HTML page versus what you present them with in a WML page can and should vary. Take a stock application, for example. On an HTML page, it would be simple, and would probably be a good idea, to display a chart showing a stock's performance over a day/week/month range along with any other data when you present a stock quote. However, the phone may or may not be able to display such a chart. If it can, the displayed image may not be large enough to be meaningful to the user.

One of the larger issues in dealing with the phone is the fact that users are not multitasking when using the browser. When you sit at your desk and you're waiting for a Web page to load, it is not likely that you have the browser taking up the whole screen. Maybe you're looking at different content in another browser window. Maybe you're reading your email, maybe you're proofreading a document, or maybe you're petting the cat. It all boils down to the fact that you're willing to put up with some degree of latency because your attention can be drawn away to other things. When users are accessing content on their phone, they are staring at the phone screen, waiting for a response. What may in quantitative terms be a short response time can seem much longer because attention is focused directly and exclusively on the device. A 10 second wait may be tolerable for an HTML page to load in a Web browser, but a 5 second wait can seem like an eternity when using a WAP-enabled phone. Keep this in mind when you decide what part of your site you are going to deliver to the user. Of course, what is acceptable varies from one application to the next, you'll have to use your judgment (and some real-world user feedback) here.

Database-Driven Applications

The key to building a compelling application that can be accessed from both a WAP-enabled phone and an HTML browser is synergy. Use the strengths of each environment to enhance the user experience, and tailor what you will deliver to the device. The beauty of the Web is its ability to deliver dynamic data that is constantly changing and unique to each user as appropriate. To this end, the information that you are going to present should live in a database. There are many free, and not free, database choices out there on every platform with drivers for any language that you want to use. For the sake of simplicity (and space), I have chosen one database and one language to present code in, but the underlying principles can be applied to any database or language.

Introduction to Employee Directory

Listing 13.1 is an example of an employee phone list that can be accessed from either an HTML browser or a WAP browser from the same URL. Although in reality there are really two different applications here, they both provide an interface into the same data store, and updates in one medium will propagate to the other. The phone book application is based on a simple SQL database running in mSQL (from Hughes Technologies at `http://www.hughes.com.au/`). Database access is accomplished through the w3-msql library that ships with mSQL. Although the code used in this chapter is specific to mSQL, the queries and principles used can be applied to any SQL database. Of course, similar code could be written in ASP, ColdFusion, Perl, JSP, and many other languages (as will be seen in the final chapters of this book).

The database that this application uses relies on one table named `emp_details`. The table has six columns: `emp_no`, `first_name`, `last_name`, `phone`, `email`, and `dept`. The `emp_no` is a sequential number that is unique and generated by the database for every row that is added to the table. The others are simply character values.

The application is accessed by the same URL from the PC browser or from the WAP browser. This is done by using a script that examines the value of the `HTTP_ACCEPT` string and then redirects the client to the appropriate code.

LISTING 13.1 An Employee Phone List

```
#!/usr/local/bin/perl

$acc = $ENV{"HTTP_ACCEPT"};
$ua = $ENV{"HTTP_USER_AGENT"};
if ($acc =~ "wml"){
        print "Location: http://delphi.phone.com/Hughes/empList.wsql\r\n\r\n";


}
else{
print" Location: http://delphi.phone.com/Hughes/empList.msql\r\n\r\n";
   print '<html>

<head>
<META HTTP-EQUIV=Refresh CONTENT="0;
➥URL=http://delphi.phone.com/Hughes/empList.msql">
</head>
<body>
</body>


</html>
';
}
```


HTML Entry Point

The entry point into this application is going to look slightly different between an HTML browser and a WAP browser for several reasons. In an HTML browser, it is safe to assume enough real estate and memory to simply display the entire phone list in a table when the user requests it. This is a very simple page, which could certainly be enhanced with a corporate banner at the top of the window and some navigation links along the left edge of the window. However, these should be kept small and relatively inconspicuous, since the employee information is what users are really interested in. This is shown in Figure 13.6.

FIGURE 13.6

We can display all rows of the phone tree in the table since neither screen size nor maximum page size are constraints on the HTML browser.

The page in Figure 13.6 is generated by a simple query into the database that looks like this:

```
    if (msqlQuery($sock,"SELECT first_name, last_name, phone, email,
➥emp_no, dept FROM emp_details ORDER BY last_name") < 0)
    {
        echo("Error : $ERRMSG\n");
        exit(1);
    }
    $res = msqlStoreResult();
    $row = msqlFetchRow($res);
>
<table border>
 <tr>
  <th>First name</th><th>Last Name</th>
➥<th>Phone</th><th>Email</th><th>Department</th>
 </tr>
<!
    while (#$row > 0)
    {
          printf("<form action=edit_emp.msql method=POST>
➥<tr><td>%s</td><td>%s</td><td>%s</td><td><a href=mailto:%s>%s</a></td>
➥<td>%s</td><td><input type=Submit name=edit value=Edit>
➥<input type=hidden name=emp_no value=%s</td></tr></form>",
```

```
        $row[0], $row[1], $row[2], urlEncode($row[3]),
➥urlEncode($row[3]), $row[5], $row[4]);
        $row = msqlFetchRow($res);
    }
    msqlFreeResult($res);
```

If we temporarily ignore everything except for the SQL statement, we can see that the application simply asks for every field from each row of the database. Stepping out a bit further we can see that there is a level of error checking built in to the application so that we can return an error from the query if one is generated (rather than causing the browser to hang or return an effectively meaningless error type 500). The results of the query are stored in a local variable `$res`, and then one row at a time is pulled out of the results and stored in an array, `$row`.

The HTML code that is used to wrap the results of the SQL query presents each row of data in a table. The entire result is iterated over, and the elements from the array are extracted and presented in a table cell. Each email address is wrapped in an `<a href="mailto:">` link, which will allow users to invoke their email client directly from their PC browser. The last column of the table contains an Edit button that will allow the user to update the record in the database (this will be discussed later on in this chapter).

Phone Entry Point

Due to screen display limitations as well as the limit on the amount of information that can be delivered to a WAP phone at one time, the entry point of the application for the WAP phone is going to be simplified to just present the list of employee names. Figure 13.7 shows a screenful of the WAP presentation.

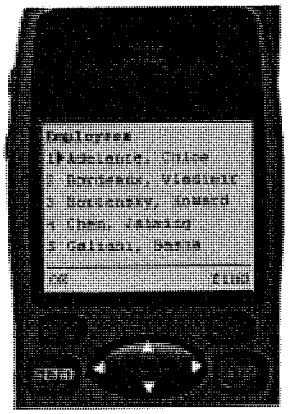


FIGURE 13.7
Even if the entire database were small enough to fit in a single deck, the table cannot be displayed in a meaningful way, so each name in the list becomes a link of its own.

In addition to the fact that only the names from the database are initially displayed because of memory limitations in WAP devices, we have to limit the number of records we can present to the user at one time. To allow access to the entire employee database, we need to keep track of which names have been delivered. Since we are only going to present nine possible records from the database in any given card, we can make the tenth item a More... link, which will present another screenful of data. To do this, we need a slightly more sophisticated select statement for our SQL select:

```
    if (msqlQuery($sock,"SELECT first_name, last_name, phone, email,
➥emp_no, dept FROM emp_details
                    WHERE last_name >= '$last_name' ORDER BY last_name") < 0)
    {
        echo("Error : $ERRMSG\n");
        exit(1);
    }
    $res = msqlStoreResult();
    $row = msqlFetchRow($res);
    echo("<select>");
    $counter = 0;
    while ((#$row > 0) && ($counter < 9))
    {
        printf("<option onpick=\"#card%s\">%s, %s</option>",
➥$row[4],$row[1],$row[0]);
        $row = msqlFetchRow($res);
        $counter++;
        if (#$row > 0){
         $last = 1;
         if (($counter % 9) == 0){
          $last = 0;
          printf("<option onpick=\"empList.wsql?last_name=%s\">
➥more...</option>",$row[1]);
          echo("</select>
            </p>
            </card>");
            }
        }
    }
    if($last == 1) {
          echo("</select>
            </p>
            </card>");
    $last = 0;
    }
```

Notice that in the SQL select statement we include a `WHERE` clause so that we can limit exactly which records are going to be selected based on a value of `last_name`. When this select statement is first called, there is no value for `last_name`, so the query will retrieve records from the start of the alphabet. The entire result of the query is stored locally in a variable, `$res`, and then read out one row at a time. A counter is used to keep track of how many rows have been read, and when the ninth record is reached, a More... link is built, with a destination of the script and the last name set as the value of the last name displayed. When a user then requests this last link, the script is re-invoked, this time with the `last_name` value assigned, ensuring that the correct record set is retrieved.

```
<option onpick="empList.wsql?last_name=Longsreth">more...</option></select>
```

The application relies on this last name value, rather than the unique identifier of `emp_no` because the value of `emp_no` is not going to be sequential according to the `last_name`. After either the More... link is built or the end of the data has been reached, the card is closed off, and the script re-iterates back over the results, building the individual cards for each record that will display the data.

Once a user selects one of the names in the list, the details associated with that user will be displayed on a card of its own. This is the only way to represent the data in a user-friendly and meaningful way. When the user selects one of the names, their details will be displayed in full on a new card. Figure 13.8 shows the details for one of the employees on the list.

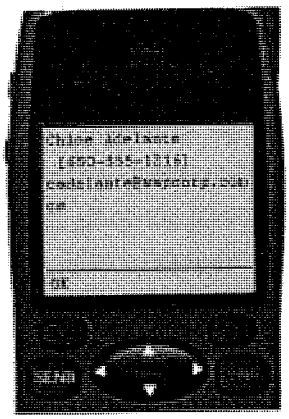


FIGURE 13.8
When the user selects one of the employee names, all of their details are displayed. The phone number is surrounded with a `wtai://wp/mc;` *so that the number is callable directly from the browser.*

Since we know that the user is going to be using this application from the phone, and it is a phone book, it only makes sense to allow the user to be able to generate a call from the results. The UP.Browser from Phone.com will allow the user to select the link that displays the phone

number, and automatically dial the number. Other devices may just present the user with a Use Number feature that will drop the phone number into the phone's voice interface, and the user can then choose to make the call. In the future, as more of the WTAI (Wireless Telephony Application Interface) specification from the WAP forum is implemented by devices, it will be possible to present the user with an option to add the information to the local phone book on his device, or maybe send a text message to the user.

Remember that in the HTML application, the email address is presented as a hyperlink with a `mailto:` action that will allow the user to launch his email application directly from his browser. Although there are WML-based email programs, there is no standard way to invoke them at this time, so the email address in the WML application is not presented as an active link.

HTML-Specific Features

The HTML application has some additional features beyond simply displaying the phone information. It allows users to either edit an existing listing or add a new listing to the database. These functions could be presented in the WML version of the application, but the entry of data from the phone is often an arduous process and should be avoided whenever possible. Providing an HTML interface into the data gives the user a much more usable way to manage the information.

Adding an Employee to the Database

An employee may be added via the Add button at the bottom of the table, and brings up a simple HTML form page as shown in Figure 13.9.

The action on this form is to post all of the field values to a script, which will then create a new row in the `emp_details` table in our database, generate an employee number for the record, and insert the data.

```
if (msqlQuery($sock, "select _seq from emp_details") < 0) {
    fatal("Query failed : $ERRMSG");
}
$res = msqlStoreResult();
$row = msqlFetchRow($res);
$sequence = (int)$row[0];
msqlFreeResult($res);

/*
** Insert the record
*/
$q = "insert into emp_details values ($sequence, '$first_name',
    '$last_name', '$dept', '$phone', '$email')";
```

```
if (msqlQuery($sock, $q) < 0) {
    fatal("Add employee failed : $ERRMSG");
}
msqlClose($sock);
```

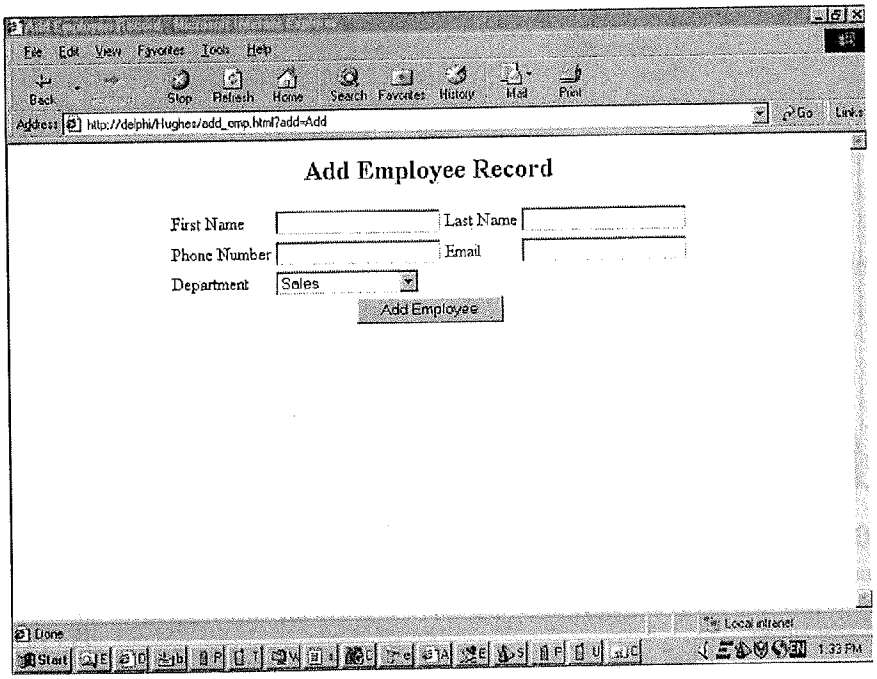


FIGURE 13.9
This HTML form allows a user to input the data. The action taken when the user presses the Add Employee button will add a new record into the database.

The `SELECT _seq` statement will generate the new sequential number that can then be used as the `emp_no` value. This value, along with all of the data that was picked up in the `POST`, is inserted into a new row in the database. When the update is completed, the user is returned to the employee list with the new record displayed in the appropriate alphabetical order.

Editing an Existing Record

The other HTML-specific feature for the database is the ability to edit an existing record. The Edit button that is presented in the last column of each row will build an edit card for the employee listed in that row, as you can see in Figure 13.10.

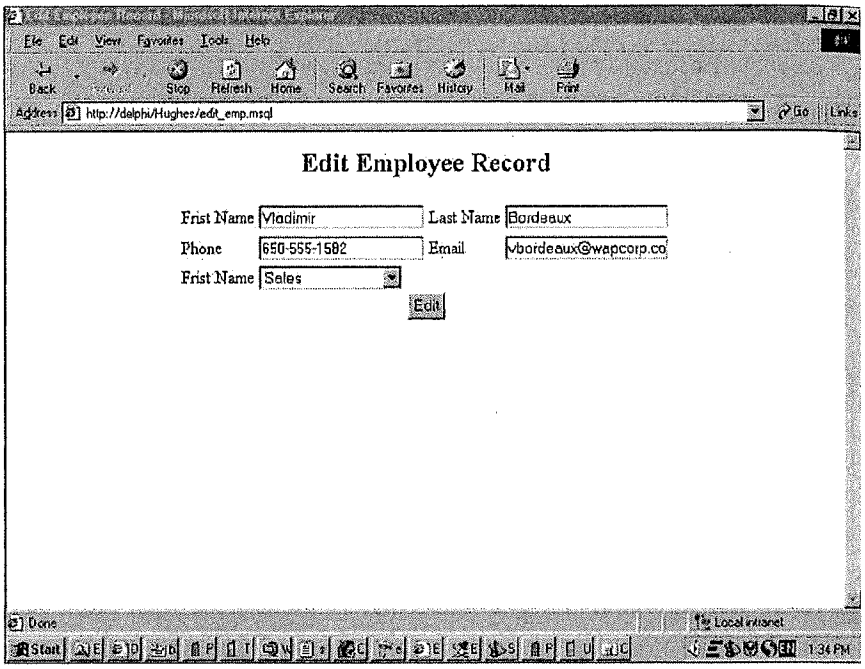


FIGURE 13.10

This HTML form allows a user to edit the data associated with a given employee. The fields in the form are automatically filled in so the user only needs to change those that apply. Changes are committed to the database as soon as the user presses the Edit button.

The edit sequence relies on the `emp_no` field in the database, which is stored in a hidden field in the form. This ensures that only the correct record is going to be updated.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
<HEAD><TITLE>Edit Employee Record</TITLE></HEAD>
<BODY>
<CENTER>
<H2>Edit Employee Record</H2>
<FORM ACTION=update_emp.msql METHOD=POST>
<TABLE align=center>
<TR>

<form action=update_emp.msql
<table>
 <tr><td>First Name</td><td><input name=first_name value=Vladimir></td>
<td>Last Name</td><td><input name=last_name value=Bordeaux></td></tr>
<tr><td>Phone</td><td><input name=phone value=650-555-1582></td>
<td>Email</td><td><input name=email value=vbordeaux@wapcorp.com></td></tr>
```

```
<td>First Name</td><td><select name=dept value=sales>
      <OPTION VALUE=sales>Sales
      <OPTION VALUE=dev>Development
      <OPTION VALUE=admin>Administration
      <OPTION VALUE=cs>Customer Support
<input type=hidden name=emp_no value=113>
</table>
<input type="submit" name="Edit" value="Edit">
</form>
</body>
</html>
```

The script that receives the post behaves very much like the script that adds a new employee to the database (this edit form should have looked familiar), however instead of performing an INSERT, it will UPDATE the record that is associated with the emp_no.

```
<!
    $emp_no = (int) $emp_no;
    if (msqlQuery($sock,"UPDATE emp_details
                    SET first_name = '$first_name', last_name = '$last_name',
                    phone = '$phone', email = '$email', dept = '$dept'
                    WHERE emp_no = $emp_no") < 0)
    {
        echo("Error : $ERRMSG\n");
        exit(1);
    }
    msqlClose($sock);
```

The int in the very first line of this code, the $emp_no value that came in with the postdata, is cast to an int so it can be effectively used in the SQL query. It must be cast this way, otherwise it will be represented as a string, and there will be a type mismatch result from the SQL statement. The UPDATE is otherwise very simple and straightforward.

Again, there is no technical limitation to prevent these add and edit functions from being presented on the phone. It is just not done here for space reasons, and also to underscore the point that data entry is simpler and cleaner from an HTML interface.

Phone-Specific Functionality

Since the phone cannot display all of the records at the same time on the initial screen, it is important to include a search capability for the database. This prevents users from having to scroll through many screens of results before finding the employee they are looking for. This feature is not really needed for the HTML application since all records are displayed on the same page, and users can use the Control-F find function from their HTML browser to search for an employee.

If you look back to Figure 13.7, you will notice the Find label bound to the options soft key. This action will bring up a find deck so the user can choose to search for a employee based on first name or last name. The searches are exclusive, meaning that they cannot search for first and last name at the same time.

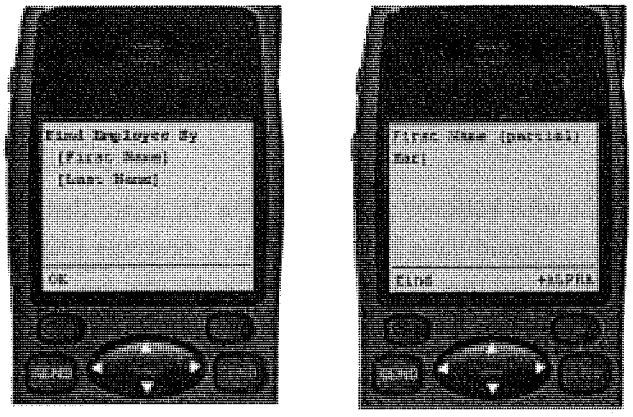


FIGURE 13.11
The WML application needs to allow the user to search for an employee based on first or last name. The user selects what he wants to search by, and a SQL query is generated.

The first or last name that is entered is then delivered on the query string to a script that queries the database for matching records. If a match is found, only the first match is returned, and if there is no match, a No Match card is returned instead. In addition to the matching record (or no results), an "all" action is bound to the options key. This will present the user with the employee listing, starting at the top of the alphabet. Following is the code for the script that runs the query and returns the results:

```
if(#$first_name >0){
    if (msqlQuery($sock,"SELECT first_name, last_name, phone, email,
➥emp_no FROM emp_details
                    WHERE first_name LIKE '%$first_name%'") < 0)
    {
        echo("Error : $ERRMSG\n");
        exit(1);
    }}
    else {
    if (msqlQuery($sock,"SELECT first_name, last_name, phone, email,
➥emp_no FROM emp_details
                    WHERE last_name LIKE '%$last_name%'") < 0)
    {
        echo("Error : $ERRMSG\n");
        exit(1);
    }}
```

```
$res = msqlStoreResult();
$row = msqlFetchRow($res);
if (#$row > 0){
    printf("%s %s<br/><a href=\"wtai://wp/mc;%s\"
➥title=\"call\">%s</a><br/>%s<br/>",
    $row[0], $row[1], $row[2],$row[2], urlEncode($row[3]));
}
else{
 echo("$first_name $last_name Not Found!");}
echo("</p>
     </card>
     </wml>");
```

The script relies on the definition of the `first_name` variable from the HTTP request to decide if the SQL query should match first or last name. This code could be enhanced to return multiple matches if they exist, but for simplicity's sake, it will only return one row of data.

Application Conclusions

The Employee Database application is just one simple possibility that can be used to present the same data to vastly different clients. The code that is used to perform this task is similar across the HTML and WML versions, but the key considerations of how much data to display at once, how to display it in a meaningful way, and how to take advantage of client-specific features are taken to heart.

Other Languages

As stated at the beginning of the chapter, the WAP Forum is on a road to convergence with the W3C, and the specifications for WML 2.0 is currently planned to be inline with XHTML (the XML version of HTML). Until this happens, it is worth noting that today WML is indeed an XML application. As XML browsers begin to permeate the market, developers can begin to move away from writing HTML and simply use XML along with Extensible Stylesheet Language (XSL).

XML tags within a document describe what the elements of the document are, rather than what they look like. The XSL does the job of describing how a given element should be displayed based on the device that is rendering the element. Take for example our employee database. Using XML, we could use tags such as `<e-address>` and `<phone-num>` to mark up the email addresses and phone numbers in the data that we present. The XSL used to build the HTML version of the application would then automatically wrap the email address in the `mailto:` anchor, and the XSL for the WAP phones would automatically build the `wtai://wp/mc;` link around the phone number so that it could be automatically called.

There are volumes on XML and XSL, and it is beyond the scope of this one chapter to explain how XML and XSL work together to transform data as appropriate to a requesting client. However, it is worth looking at the Cocoon project from `http://xml.apache.org/cocoon`. This project (which is currently under way) provides a way to accomplish exactly what is discussed in this chapter, including employing the HTTP headers that are delivered by a device request to ensure that the correct XSL information is used for a given device. This allows content to be repurposed appropriately, allowing the support of a wide range of device and user agents.

Summary

This chapter has taken a quick look at what it means to deliver the same content to an HTML browser as well as a WAP browser. The key message is that while HTML can be delivered to a phone through a translator, taking the time to write a WML interface into an application is well worth the effort in terms of both usability and functionality. Although WML and HTML are on the road to convergence with the forthcoming XHTML specification from the W3C, it will still be some time before the specifications become reality. In preparing yourself for that time, and even in preparing yourself to build a scalable Web application, content and data should be abstracted from the presentation of that content. The simplest and most effective way to do this is to store your data in a database.

Any programming language can be used to provide access to the same content base to any client. The key is sticking with what you know and what you are familiar with. It looks as though XML/XSL will play a huge role in content delivery to a wide variety of clients in the future. Abstracting your presentation from your content now will get you a long way toward the future.

E-Commerce

CHAPTER

18

IN THIS CHAPTER

E-Commerce Application Overview

For this fourth and final sample application, we are going to take an in-depth look at how e-commerce might work with a WAP device. In this example, we will revisit our fictitious store, Burgerworld.com—the one stop shop for hungry computer professionals, which sells burgers, fries, and computers. (A great business model, if you ask us!)

While shopping with a mobile device may be unfamiliar to many people, the Internet has already carved out models for electronic shopping that consumers have grown accustomed to and even expect. The backbone of an e-commerce shopping experience is the virtual shopping cart. The cart we'll create in this example represents a session that you might use to track a user's login process through a content management system.

At this point, we would like to note that this application provides the basic functionality of an e-commerce application, to demonstrate the fundamental techniques involved in designing a WML interface that operates with a data-driven backend for e-commerce. This example does not cover security or credit-card validation and therefore would not be recommended for real-world deployment without those components.

This sample application will be more technologically challenging than our previous ones. We will be coding this example in JSP, using TOMCAT or JRun servers.

NOTE

Java Server Pages (JSP) is a scripting capability for Web pages that allows Java as well as a few special tags to be embedded into a Web file (HTML/XML, and so on). JSP/ASP facilitate dynamic Web-page delivery to client applications.

E-Commerce Application Walkthrough

We'll begin by conducting a step-by-step walkthrough of the code contained in this application. If you'd like to see any of the sections we'll discuss here in their complete context, please reference the end of the chapter, where all of the code listings are given in their entirety.

Keeping Track of the Data

In this application, we use several kinds of data, and each time we need to access one, we are interfacing into a database using JDBC to a data source called "shop." This database will drive our WAP application. It includes the product catalog, shopping cart storage, and ordering information. Refer to Chapter 17, "Scheduling," for database setup parameters.

The database itself has five tables. The first one is really our session management. In this example, there is a login screen that needs to be processed. For this application, the login page will serve as our session manager. When a user reaches the `login.jsp` page (see Listing 18.1), the application will query a table called `Vars` to determine what shopping cart a user will have access to. You can see this table in Figure 18.1. From this point on, we pass this cart identification with the user to make sure we can track his session.

Vars : Table

	Value	Name
▶	22	CART
*	0	

FIGURE 18.1
The `Vars` table contains shopping cart identification information.

A second table, called `Items`, is at the core of our application. It is shown in Figure 18.2. In this table we have a particular item, stock keeping units (SKUs—which are unique product numbers used in inventory tracking) for an item, a category, and pricing information. In a more complete application, this table might have inventory information or other fulfillment information.

Items : Table

	Item_Category	Item_Name	Item_Cost	Item_SKU	Item_Shipping	Item_ID
▶ ⊞	Computers	AS400	$22,000.00	16521	$205.00	1
⊞	Computers	PDP11	$4.60	16502	$300.00	2
⊞	Computers	PDP8	$3.00	19295	$275.00	3
⊞	Soda	Mountain Dew	$2.00	743456	$0.01	4
⊞	Soda	7 Up	$2.00	7668745	$0.01	5
⊞	Soda	Sprite	$2.00	8454590	$0.01	6
⊞	Soda	Dr Pepper	$2.00	1224356	$0.01	7
*			$0.00		$0.00	(AutoNumber)

FIGURE 18.2
The `Items` table contains pricing, category, and SKU information on various products.

The `ShoppingCart_Items` table stores all the data we need for cart management. In this table, we are tracking a shopping cart ID, which will be unique for each user session; the items in a particular cart (based upon `Item_ID` numbers from the `Items` table); and quantities. You can see what this table looks like in Figure 18.3.

Finally, the last two tables contain our data for a completed order. When a user "checks out" and then enters his order information, the completed order information is entered into the `Orders` table, seen in Figure 18.4, and corresponding entries for the ordered items are deposited into the `Orders_Products` table, seen in Figure 18.5.

ShoppingCartID	Item	Quantity	Shop_Item_ID
			1
2	1	2	2
18	2	2	3
18	5	2	4
19	2	2	5
19	2	2	6
20	2	2	7
0	0	0	(AutoNumber)

FIGURE 18.3
The `ShoppingCart_Items` *table contains shopping cart ID numbers, items, and quantities.*

Username	Address	Phone	CreditCard	OrderID
WEB	null	555-1212	null	20
				0

FIGURE 18.4
The `Orders` *table contains user information for a complete order, such as address, phone number, and order ID number.*

OrderID	ItemID	ItemIndex	Quantity
20	2	1	2
0	0	(AutoNumber)	0

FIGURE 18.5
The `Orders_Products` *table contains the confirmed list of products the customer ordered.*

Preparing the Header, Data Source, and Session ID

Now that we have discussed the data model, let's start walking through the heart of this example with `login.jsp`, from Listing 18.1. We are now coding in JSP, so our header information needs to change a little bit because of the response type. Therefore, the header for each JSP file will contain the standard XML directive, but additionally include the following line:

```
<% response.setContentType("text/vnd.wap.wml"); %>
```

This instructs the server that the mime-type will be `text/vnd.wap.wml`, which is the accepted WML directive.

After we have established that, we prime our database engine, and query the database for a variable named `CART` from the `Vars` table. The goal here is to select a unique cart ID that we can establish for the user.

```
<%@ page import="java.sql.* >

<%
    Connection dbConn = null;
    int CartID=-1;
    ResultSet get_cart = null;

    try
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        dbConn = DriverManager.getConnection(
            "jdbc:odbc:shop"
            );
        Statement userStmt = dbConn.createStatement();
        get_cart = userStmt.executeQuery(
            "SELECT VALUE from Vars WHERE Name='CART'"
        );

        if (get_cart.next()) {
               CartID = get_cart.getInt("Value");
            userStmt.executeUpdate(
         "UPDATE Vars SET [Value]=[Value]+1 WHERE Name='CART'"
            );
        }
        else
        {
         CartID=1;
         userStmt.executeUpdate(
         "INSERT INTO Vars(Name, [Value]) SELECT 'CART' AS EXPR1, 2 as EXPR2"
         );
        }


    }
    catch(Exception e)
    {
        out.println("Exception!!!");
        out.println(e.getMessage());
        return;
    }

%>
```


In the preceding code sample, we made provisions for error checking to ensure that the Vars table contains the correct entry if you are installing this application for the first time. When the code is executed, the server will first query the table for a value, and if a value is found, it will

then increment the value by executing an update statement that sets the `[value] = [value] + 1`. If the value is not found, we insert the variables into the `Vars` table and set the `CartID=1` to prime the session variable.

TIP

When doing session management, two options exist for WAP devices: a cookie or a session variable. We chose a session variable because some of our tests yielded errors when we used cookies.

NOTE

The database we used for our implementation was a Microsoft Access database. As a result, the column `value` requires brackets because it is a reserved word. If you have ported this example into a different database, those brackets may be unnecessary and could cause an error.

After we complete the session variable as illustrated above, we can construct our WAP deck.

Creating a Login that Tracks the User

In this application, our `login.jsp` file will serve as a welcome. We give a quick advertisement and then give the user an option to go to the next card, but pass our unique `CartID` in the `url` parameters. We have now begun to track our user!

TIP

This card would be a great candidate for a timer. For the sake of simplicity, we chose to eliminate the timer, but you could create one of your own. For more information on timers, see Chapter 10, "Using Timers."

The following code snippet from Listing 18.1 shows the first WML card of the application.

```
<wml>
    <card id="Welcome" title="Welcome">
    <p>
        <b>Burgerworld.com</b>
        Burgers, Fries, Soda, and Computers... to go<br/>
        <a href="main.jsp?CartID=<%=CartID%>">Enter</a>
```

```
    </p>
    </card>
</wml>
```

When the user selects the hyperlink labeled `Enter`, he will navigate to the `main.jsp` file, passing his `CartID` value to that deck. You can see what this looks like in Figure 18.6.

FIGURE 18.6
The Burgerworld login screen shows the first card, where we start to keep track of our user's ID.

Dynamically Generating Product Catalogs

Now that the user has logged in, it is time to present him with the product catalog. In our case, we are going to dynamically generate a deck for item categories and the cards within those categories as well.

The application will select all of the categories from the database and present them as main menu choices. Then, under each category, the application will construct a card that will serve as the submenu. In this example, the submenus only go one level deep; a category can only contain items and cannot contain another category.

To accomplish this, we are going to need a few different database queries. At the top level, we will query the database for the `DISTINCT item_category` information from the `Items` table. This will form the main menu as seen in the following code sample, taken from Listing 18.2.

```
<%
    Connection dbConn = null;
    ResultSet get_cat = null;
    ResultSet get_products = null;
    boolean isMore = false;
```

```
// get a connection to the database
try
{
    // instantiate the db driver
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    // setup the connection
    dbConn = DriverManager.getConnection(
        "jdbc:odbc:shop"
        );

    // create a statement on the connection
    Statement userStmt = dbConn.createStatement();

    // issue the SQL statements
    get_cat = userStmt.executeQuery(
        "SELECT Distinct Item_Category from Items"
    );


}
catch(Exception e)
{
    out.println("Exception!!!");
    out.println(e.getMessage());
    return;
    // assume no error conditions, for now
}

%>
```

TIP

Dynamically generating a catalog promotes a very scalable architecture for the application. If we were to add a new category of items to the table, the category would appear on our WAP device when the user looked at the category card. Moreover, if we dynamically change the item information in any way, the entire menu will restructure itself.

CAUTION

The method used above does present some scalability issues. If the site is a higher-volume site, the application should take advantage of cached-queries or WAP browser cache management by expiring pages or sending cache notifications in order to minimize database load.

Now that we have the main query, called `get_cat`, which contains all of the unique categories, we can construct the main menu by cycling through the list of unique categories as shown in the following code sample taken from Listing 18.2. You can see what this card looks like in Figure 18.7.

```
<wml>
    <card id="MainMenu" title="Menu">
        <p>
            Burgerworld.com<br/>
            <%
                try
                    {
                  while (get_cat.next())
                  {
                            String Category =
➥get_cat.getString("Item_Category");
                %>
                <a href="#<%=Category%>"><%= Category %></a><br/>
                <%
                        }
                    }
                catch(SQLException e)
                    {
                        }
            %>
            </p>
    </card>
```

NOTE

You'll notice in the preceding code that all links are going to a local reference called `Category`. The card with id `Category` will be dynamically generated with the remainder of the code.

FIGURE 18.7
The menu, listing `Computers` *and* `Soda`*, has been dynamically generated from the data source.*

Believe it or not, that was probably the easy part. Now, we'll repeat the preceding step, and dynamically generate product cards in the process. To do this, we will repeat the exact same query for the distinct categories, but then create a card for each one with an ID called `Category` as seen in the following code sample, taken from Listing 18.2.

```
<%
    ResultSet get_catcard = null;
    ResultSet get_Items = null;
    try
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        dbConn = DriverManager.getConnection(
            "jdbc:odbc:shop"
            );
        Statement userStmt = dbConn.createStatement();
        get_catcard = userStmt.executeQuery(
            "SELECT Distinct Item_Category from Items"
        );
    }
    catch(Exception e)
    {
        out.println("Exception!!!");
        out.println(e.getMessage());
        return;
    }
```

The query `get_catcard` generates a list of items in a particular category. It does this by retrieving items from the `Items` table whose category matches the one currently in use. The query keeps looping through the different categories, generating sublists of products, until all of the

categories in the table have been queried. Assume the query `get_catcard` is looking at the category `Sodas`. It will retrieve all of the products in the `Items` table that are categorized as sodas. Once the query has retrieved all of the soda products, it stops, goes back, and looks for the next category. When the application finds the `Burgers` category, it will reinitiate the `get_catcard` query, finding all the different burger products in the `Items` table. All of those burger products will go into a sublist, which will link each product name to another location where the user can learn more about the product. You can see how we did this in the following code sample, taken from Listing 18.2.

```
    try
     {
        while (get_catcard.next())
         {
            String CardCategory = get_catcard.getString("Item_Category");

             try
                    {
                    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                    dbConn = DriverManager.getConnection(
                        "jdbc:odbc:shop"
                        );
                    Statement userStmt = dbConn.createStatement();
                    get_Items = userStmt.executeQuery(
                       "SELECT * FROM Items WHERE Item_Category='"
➥+ CardCategory + "'"
                        );
                }
                catch(Exception e)
                {
                    out.println("Exception!!!");
                    out.println(e.getMessage());
                    return;
                }

%>
```

The next set of code is a little more difficult to understand. Keep in mind that in the code execution cycle, we are looping through distinct categories at this point.

Now, we are going to dynamically generate a card. The important thing to note is that the card ID corresponds to the `CardCategory`. Since the main menu links are linking to a particular category, we will dynamically generate cards by using the distinct category names as the card IDs, as seen in the following code sample. The card for the category `Computers` can be seen in Figure 18.8.

```
        <card id="<%= CardCategory %>" title="Order <%= CardCategory%>">
              <p>
                   <b><%= CardCategory %></b><br/>
<%
                 while (get_Items.next())
                 {
%>
                   <a href="orderproduct.jsp?CartID=<%=request.getParameter
➥("CartID")%>&amp;ItemID=<%=get_Items.getString("Item_ID")%>">
➥<%=get_Items.getString("Item_Name")%></a><br/>
<%                 } %>
              </p>
         </card>
<%    }
      }
    catch (SQLException e)
      {
      }
%>
</wml>
```

FIGURE 18.8
The `Computers` *category's submenu is displayed on this dynamically generated card.*

CAUTION

In the preceding example, we are dynamically generating an entire WAP deck. This methodology does have its risks. If we are not careful, the deck could grow to a size that exceeds a device's ability to publish it, and slow down the application. If the virtual store had many more items, it might make more sense to publish the main menu deck as a single main menu card and then have each card be dynamic but self-contained.

s we move on to the ordering stage of the application, we need to give the application some eans by which to track an order's components. One other important thing to note in the pre-ding code sample is that we are tracking items by their unique Item ID. This will become ven more apparent and critical when we order a product.

NOTE

When writing an e-commerce application, keeping items unique is essential from a business fulfillment perspective, but also functional from an application development perspective. Passing around complete item names could make it difficult to keep track of a long list of products.

ach product in the previous example contained a link to another WML deck called derproduct.jsp. We could have again added this card to the `main.jsp` deck, but because of ze limitations and presentation issues, we elected to make it another deck altogether.

TIP

When creating dynamic WML decks, it is occasionally necessary to separate decks based upon your database queries. In general, if a page is drilling down into a specific item's detail, it is not a good idea to put it in the same deck that has basic information due to query restrictions.

howing and Storing Dynamic Product Information

ur next deck, `orderproduct.jsp`, is designed to provide the user with details about a partic-lar product. In this deck we will show all of the relevant information about a particular prod-ct, and then allow the user to enter a quantity. This deck is entered via the `main.jsp` deck, here a user selects a category and a product. When the user arrives at the `orderproduct.jsp` eck, he will have passed the `Item_ID` for the product he wants from the `Items` table as a URL ariable named `ItemID`.

o start this deck off, we are going to do a `SELECT *` query against the `Items` table to retrieve ll of the item information for the selected item, as shown in the code sample below, taken om Listing 18.3.

```
%
   Connection dbConn = null;
   ResultSet get_Item = null;
```

```
    // get a connection to the database
    try
    {
        // instantiate the db driver
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        // setup the connection
        dbConn = DriverManager.getConnection(
            "jdbc:odbc:shop"
            );

        // create a statement on the connection
        Statement userStmt = dbConn.createStatement();

        // issue the SQL statements
        get_Item = userStmt.executeQuery(
            "SELECT * from Items where Item_ID="
➥ + request.getParameter("ItemID")
        );

    }
    catch(Exception e)
    {
        out.println("Exception!!!");
        out.println(e.getMessage());
        return;
    }

%>
```

After we complete our query, we will display the data and ask the user to enter a quantity for the item. Additionally, we have established a soft key called `Order` that will pass along the `CartID` parameters and `ItemID` parameter via a form using the standard `postfield` method. This is shown in the following code sample and in Figures 18.9 and 18.10.

```
<wml>
    <card id="OrderIt" title="Order Product">
         <do type="accept" label="Order">
            <go href="addtocart.jsp">
              <postfield name="Quantity" value="$Quantity"/>
              <postfield name="CartID" value="<%= request.getParameter
➥("CartID")%>"/>
              <postfield name="ItemID" value="<%= request.getParameter
➥("ItemID")%>"/>
            </go>
         </do>
        <p>
<%   try
```

```
{
        if (get_Item.next())
        {
%>
       <b><%= get_Item.getString("Item_Name")%></b><br/>
       Cost <%= get_Item.getString("Item_Cost")%><br/>
       SKU #<%= get_Item.getString("Item_SKU")%><br/>
       Quantity : <input type="text" name="Quantity" format="N*"/>

<%              }
      }
      catch (Exception e)
      {
      }
%>
         </p>
 </card>

</wml>
```

FIGURE 18.9
This card shows the user a dynamically generated list of products in the `Computer` *category.*

TIP

By using the `format="N*"` in the `<input>` tag for the quantity, we are forcing the user to enter at least one number before the form can be submitted. Therefore, this serves as an excellent way to verify data. When there is no data entered, the device will not offer the user the `Order` soft key. As soon as he enters a number, the WAP device will present him with the `Order` soft key as the data has been validated into the `N*` format.

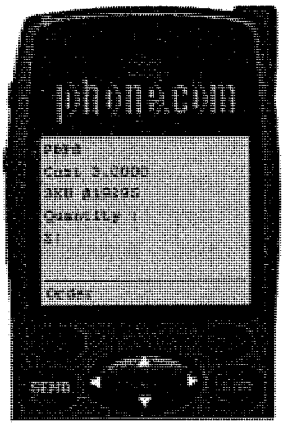


FIGURE 18.10
This card asks what quantity of the item the user would like to purchase.

Now that the user has entered a quantity as well as verified that he is ordering the correct product, we need to send this data to a deck that will place the item in his cart. As you saw earlier, when the user enters a quantity and presses the `Order` soft key, we navigate to a deck called `addtocart.jsp`. The `addtocart.jsp` file takes the item identification number and cart identification number, and places it into the `Shoppingcart_Items` table. This is the table where we store the ordered products until the user checks out.

TIP

WAP is like any other Internet connection method; a user could theoretically shut off his WAP device at any point and "cancel" his session. Therefore, it's always a good idea to store data at multiple stages of the ordering process, in case a user aborts his session using a non-traditional exit strategy. It should further be noted that the storage of certain data may, in some countries, run counter to data protection legislation. You should ensure that your applications comply with the appropriate legislation for the region where it will be deployed.

The following code example, from Listing 18.3, shows how the application places product information into the user's shopping cart.

```
<%
    Connection dbConn = null;
    ResultSet get_Item = null;

    // get a connection to the database
    try
    {
```

```
        // instantiate the db driver
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        // setup the connection
        dbConn = DriverManager.getConnection(
            "jdbc:odbc:shop"
            );

        // create a statement on the connection
        Statement userStmt = dbConn.createStatement();

        // issue the SQL statements
        userStmt.executeUpdate(
            "INSERT INTO ShoppingCart_Items(ShoppingCartID, Item, Quantity)
➥SELECT " + request.getParameter("CartID") + " AS EXPR1, " +
➥request.getParameter("ItemID") + " as EXPR2, "
➥+ request.getParameter("Quantity")
        );


    }
    catch(Exception e)
    {
        out.println("Exception!!!");
        out.println(e.getMessage());
        return;
        // assume no error conditions, for now
    }

%>
```


Creating a "Checkout" Procedure

The user has now logged in and has an active cart, has chosen a category, picked an item, and entered a quantity. Now, we must offer him a choice to continue shopping or to check out and purchase the items currently stored in his shopping cart. Since we are tracking the user session via a `CartID`, we just need to pass the `CartID` back to the `main.jsp` page to allow the user to continue shopping. If he would rather check out, we then need to route him to the `checkout.jsp` deck. The following code sample, taken from Listing 18.3, does this. You can see what this card looks like in Figure 18.11.

```
<wml>
    <card id="OrderIt" title="Order Product">
        <p>
            <b>The item is in your cart!</b><br/>
                <a href="main.jsp?CartID=<%=request.getParameter
➥("CartID")%>">Shop More</a>
```

```
            <a href="checkout.jsp?CartID=<%=request.getParameter
➥("CartID")%>">Check Out</a>
        </p>
    </card>

</wml>
```

FIGURE 18.11
This card confirms the order and offers the user the option to continue shopping or move on to final checkout.

The `checkout.jsp` deck is designed to allow a user to verify his order and enter his order information, address information, and credit card data. The page then takes the user to the final deck, called `finishorder.jsp`. The following section, taken from Listing 18.3, executes a simple query against the database to retrieve the items that are in a cart and then displays the data in an intuitive format.

```
<%
    Connection dbConn = null;
    ResultSet getCart = null;

    // get a connection to the database
    try
    {
        // instantiate the db driver
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        // setup the connection
        dbConn = DriverManager.getConnection(
            "jdbc:odbc:shop"
            );
```

```
        // create a statement on the connection
        Statement userStmt = dbConn.createStatement();

        // issue the SQL statements
        getCart = userStmt.executeQuery(
            "SELECT * FROM ShoppingCart_Items INNER JOIN Items ON ShoppingCart
➥_Items.Item=Items.Item_ID WHERE ShoppingCartID=" +
➥request.getParameter("CartID")
        );


    }
    catch(Exception e)
    {
        out.println("Exception!!!");
        out.println(e.getMessage());
        return;
    }

%>

<wml>
    <card id="Checkout" title="Checkout">
        <p>
        <b>Your order:</b><br/>

<%   try
        {
            while (getCart.next())
            {
%>
                <%=getCart.getString("Quantity")%> :
➥<%= getCart.getString("Item_Name")%><br/>
<%
                }
        }
        catch (Exception e)
        {
        }

%>
            <a href="#add">Continue</a>
        </p>
    </card>
```

```
<card id="add" title="Add an appointment">
    <do type="accept" label="Add">
     <go href="finishorder.jsp">
       <postfield name="address" value="$Address"/>
       <postfield name="creditcard" value="$CreditCard"/>
       <postfield name="CartID" value="<%= request.getParameter("CartID")%>"/>
     </go>
    </do>

  <p>
    <b>Order Information</b><br/>
    Address: <input type="text" name="Address"/>
    Credit Card: <input type="text" name="CreditCard" />
  </p>
</card>

</wml>
```

You can see the order verification that occurred in the preceding code sample in Figure 18.12. The address entry field is shown in Figure 18.13 and the credit card number entry field is shown in Figure 18.14.

FIGURE 18.12
This card confirms with the user which products he is about to buy.

As noted earlier, this application does not simulate any credit card verification; but such verification would be required in a real-world application.

The `finishorder.jsp` deck is really more of a functional deck in terms of data marshalling than any of our past ones. Since we are using the `ShoppingCart_Items` table as an order staging zone, the user has now confirmed his order and we need to send that order to our permanent order tables, called `Orders` and `Order_Items`.

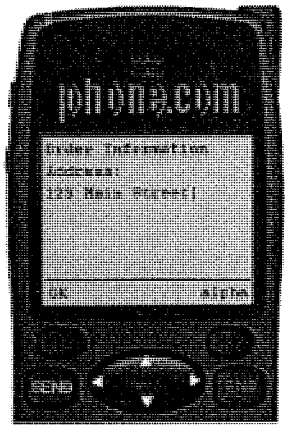


FIGURE 18.13
This card allows the user to enter his address, which is stored in the `Orders` *table.*

FIGURE 18.14
Scrolling down on the same card, users can enter their credit card information, which is also stored in the `Orders` *table.*

As we mentioned, the `Order's` table will hold all of the order information: name, credit card, and address. In the following code, we execute two SQL statements. The first creates the new order, and the second copies the cart's contents from the `ShoppingCart_Items` table to the new table called `Order_Items`.

```
<%
    Connection dbConn = null;

    // get a connection to the database
    try
    {
```

```
        // instantiate the db driver
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        // setup the connection
        dbConn = DriverManager.getConnection(
            "jdbc:odbc:shop"
            );

        // create a statement on the connection
        Statement userStmt = dbConn.createStatement();

        // issue the SQL statements
        userStmt.executeUpdate(
            "INSERT INTO Orders(Username, Address, Phone, CreditCard,
➥OrderID) SELECT 'WEB' as EXPR1, '" + request.getParameter("Address") +
➥ "' AS EXPR2, '555-1212' as EXPR3, '" + request.getParameter("CreditCard")
➥ + "' as expr4, " + request.getParameter("CartID") + " as expr5"
        );

        userStmt.executeUpdate(
            "INSERT INTO Orders_Products(OrderID, ItemID, Quantity)
➥ SELECT ShoppingCartID, Item, Quantity FROM ShoppingCart_Items WHERE
➥ ShoppingCartID=" + request.getParameter("CartID")
        );


    }
    catch(Exception e)
    {
        out.println("Exception!!!");
        out.println(e.getMessage());
        return;
        // assume no error conditions, for now
    }

%>
```

Finally, we thank the user for his order, and offer him the option to continue to shop. You can see what this card looks like in Figure 18.15.

```
<wml>
    <card id="Notify" title="NotifyUser">
        <p>
            <b>Thanks for your order!</b><br/>
            <a href="login.jsp">Place a new order</a>
        </p>
    </card>
</wml>
```

FIGURE 18.15
This card thanks the user for his order, and offers him the option of placing a new order.

Complete Code Listings

Now that we've see how the code works and how to build shopping-cart functionality with WAP, you can reference the complete code listings in the following section. The code listings can also be found on the CD-ROM that accompanies this book.

login.jsp

LISTING 18.1 The `login.jsp` File Creates the Login Page

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">
<% response.setContentType("text/vnd.wap.wml"); %>
<%@ page import="java.sql.*, javax.servlet.http.Cookie" %>

<%
    Connection dbConn = null;
    int CartID=-1;
    ResultSet get_cart = null;

    try
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        dbConn = DriverManager.getConnection(
            "jdbc:odbc:shop"
            );
        Statement userStmt = dbConn.createStatement();
```

LISTING 18.1 Continued

```
        get_cart = userStmt.executeQuery(
            "SELECT VALUE from Vars WHERE Name='CART'"
        );

        if (get_cart.next()) {
               CartID = get_cart.getInt("Value");
            userStmt.executeUpdate(
            "UPDATE Vars SET [Value]=[Value]+1 WHERE Name='CART'"
            );
        }
        else
        {
           CartID=1;
           userStmt.executeUpdate(
           "INSERT INTO Vars(Name, [Value]) SELECT 'CART' AS EXPR1, 2 as EXPR2"
           );
        }




    }
    catch(Exception e)
    {
        out.println("Exception!!!");
        out.println(e.getMessage());
        return;
        // assume no error conditions, for now
    }

%>

<wml>
    <card id="Welcome" title="Welcome">
    <p>
        <b>Burgerworld.com</b>
        Burgers, Fries, Soda, and Computers... to go<br/>
        <a href="main.jsp?CartID=<%=CartID%>">Enter</a>
    </p>
    </card>
</wml>
```

ain.jsp

ISTING 18.2 The main.jsp File Drives Dynamic Catalog Content to the User

```
% response.setContentType("text/vnd.wap.wml"); %>
?xml version="1.0" encoding="UTF-8"?>
!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
http://www.wapforum.org/DTD/wml12.dtd">
%@ page import="java.sql.*, javax.servlet.http.Cookie" %>

%
   Connection dbConn = null;
   ResultSet get_cat = null;
   ResultSet get_products = null;
   boolean isMore = false;


   // get a connection to the database
   try
   {
       // instantiate the db driver
       Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

       // setup the connection
       dbConn = DriverManager.getConnection(
           "jdbc:odbc:shop"
           );

       // create a statement on the connection
       Statement userStmt = dbConn.createStatement();

       // issue the SQL statements
       get_cat = userStmt.executeQuery(
           "SELECT Distinct Item_Category from Items"
       );


   }
   catch(Exception e)
   {
       out.println("Exception!!!");
       out.println(e.getMessage());
       return;
       // assume no error conditions, for now
   }
```

LISTING 18.2 Continued

```
%>

<wml>
    <card id="MainMenu" title="Menu">
        <p>
            Burgerworld.com<br/>
            <%
                try
                    {
                    while (get_cat.next())
                            {
                            String Category = get_cat.getString
➥("Item_Category");
                %>
                <a href="#<%=Category%>"><%= Category %></a><br/>
                <%
                        }
                    }
                catch(SQLException e)
                    {
                    }
            %>
            </p>
    </card>

<%
    ResultSet get_catcard = null;
    ResultSet get_Items = null;
    try
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        dbConn = DriverManager.getConnection(
            "jdbc:odbc:shop"
            );
        Statement userStmt = dbConn.createStatement();
        get_catcard = userStmt.executeQuery(
            "SELECT Distinct Item_Category from Items"
        );
    }
    catch(Exception e)
    {
    out.println("Exception!!!");
    out.println(e.getMessage());
        return;
```

```
}

try
  {
    while (get_catcard.next())
        {
        String CardCategory = get_catcard.getString("Item_Category");

          try
                {
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                dbConn = DriverManager.getConnection(
                    "jdbc:odbc:shop"
                    );
                Statement userStmt = dbConn.createStatement();
                get_Items = userStmt.executeQuery(
                    "SELECT * FROM Items WHERE Item_Category='"
➥+ CardCategory + "'"
                    );
              }
              catch(Exception e)
              {
                  out.println("Exception!!!");
                  out.println(e.getMessage());
                  return;
              }

%>
              <card id="<%= CardCategory %>" title="Order <%=
➥CardCategory%>">
                  <p>
                 <b><%= CardCategory %></b><br/>
<%
              while (get_Items.next())
              {
%>
      <a href="orderproduct.jsp?CartID=<%=request.getParameter("CartID")
➥%>&amp;ItemID=<%=get_Items.getString("Item_ID")%>"><%=get_Items.getString
➥("Item_Name")%></a><br/>
<%              } %>
                </p>
              </card>
<%    }
      }
   catch (SQLException e)
       {
```

LISTING 18.2 Continued

```
        }
%>
</wml>
```

orderproduct.jsp

LISTING 18.3 The `orderproduct.jsp` File Allows the User to Verify the Item and Specify a Quantity

```
<% response.setContentType("text/vnd.wap.wml"); %>
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">
<%@ page import="java.sql.*, javax.servlet.http.Cookie" %>

<%
    Connection dbConn = null;
    ResultSet get_Item = null;

    // get a connection to the database
    try
    {
        // instantiate the db driver
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        // setup the connection
        dbConn = DriverManager.getConnection(
            "jdbc:odbc:shop"
            );

        // create a statement on the connection
        Statement userStmt = dbConn.createStatement();

        // issue the SQL statements
        get_Item = userStmt.executeQuery(
            "SELECT * from Items where Item_ID=" +
➥request.getParameter("ItemID")
        );


    }
    catch(Exception e)
    {
```

```
        out.println("Exception!!!");
        out.println(e.getMessage());
        return;
        // assume no error conditions, for now
    }

%>

<wml>
    <card id="OrderIt" title="Order Product">
         <do type="accept" label="Order">
            <go href="addtocart.jsp">
              <postfield name="Quantity" value="$Quantity"/>
              <postfield name="CartID" value="<%=
➥request.getParameter("CartID")%>"/>
              <postfield name="ItemID" value="<%=
➥request.getParameter("ItemID")%>"/>
            </go>
         </do>
         <p>
<%    try
    {
            if (get_Item.next())
            {
%>
          <b><%= get_Item.getString("Item_Name")%></b><br/>
          Cost <%= get_Item.getString("Item_Cost")%><br/>
          SKU #<%= get_Item.getString("Item_SKU")%><br/>
          Quantity : <input type="text" name="Quantity" format="N*"/>

<%              }
        }
        catch (Exception e)
        {
        }
%>
            </p>
    </card>

</wml>
```

addtocart.jsp

LISTING 18.4 The addtocart.jsp File Adds the Selected Item to the User's Cart and Prompts for "Checkout"

```
<% response.setContentType("text/vnd.wap.wml"); %>
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">
<%@ page import="java.sql.*, javax.servlet.http.Cookie" %>

<%
    Connection dbConn = null;
    ResultSet get_Item = null;

    // get a connection to the database
    try
    {
        // instantiate the db driver
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        // setup the connection
        dbConn = DriverManager.getConnection(
            "jdbc:odbc:shop"
            );

        // create a statement on the connection
        Statement userStmt = dbConn.createStatement();

        // issue the SQL statements
        userStmt.executeUpdate(
            "INSERT INTO ShoppingCart_Items(ShoppingCartID, Item, Quantity)
➥ SELECT " + request.getParameter("CartID") + " AS EXPR1, "
➥ + request.getParameter("ItemID") + " as EXPR2, " + request.getParameter
➥("Quantity")
        );


    }
    catch(Exception e)
    {
        out.println("Exception!!!");
        out.println(e.getMessage());
        return;
        // assume no error conditions, for now
    }
```

```
%>

<wml>
   <card id="OrderIt" title="Order Product">
       <p>
           <b>The item is in your cart!</b><br/>
             <a href="main.jsp?CartID=<%=request.getParameter
➥("CartID")%>">Shop More</a>
             <a href="checkout.jsp?CartID=<%=request.getParameter
➥("CartID")%>">Check Out</a>
       </p>
   </card>

</wml>
```

checkout.jsp

LISTING 18.5 The checkout.jsp File Lists All Ordered Items and Allows the User to Enter His Shipping and Ordering Data

```
<% response.setContentType("text/vnd.wap.wml"); %>
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">
<%@ page import="java.sql.*, javax.servlet.http.Cookie" %>

<%
    Connection dbConn = null;
    ResultSet getCart = null;

    // get a connection to the database
    try
    {
        // instantiate the db driver
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        // setup the connection
        dbConn = DriverManager.getConnection(
            "jdbc:odbc:shop"
            );

        // create a statement on the connection
        Statement userStmt = dbConn.createStatement();
```

LISTING 18.5 Continued

```
        // issue the SQL statements
        getCart = userStmt.executeQuery(
            "SELECT * FROM ShoppingCart_Items INNER JOIN Items
➥ON ShoppingCart_Items.Item=Items.Item_ID WHERE ShoppingCartID="
➥+ request.getParameter("CartID")
        );


    }
    catch(Exception e)
    {
    out.println("Exception!!!");
    out.println(e.getMessage());
    return;
    // assume no error conditions, for now
    }

%>

<wml>
    <card id="Checkout" title="Checkout">
    <p>
        <b>Your order:</b><br/>

<%  try
    {
        while (getCart.next())
    {
%>
                <%=getCart.getString("Quantity")%> :
➥<%= getCart.getString("Item_Name")%><br/>
<%
            }
        }
        catch (Exception e)
        {
        }

%>
            <a href="#add">Continue</a>
        </p>
    </card>

<card id="add" title="Add an appointment">
```

```
<do type="accept" label="Add">
    <go href="finishorder.jsp">
      <postfield name="address" value="$Address"/>
      <postfield name="creditcard" value="$CreditCard"/>
      <postfield name="CartID" value="<%= request.getParameter("CartID")%>"/>
   </go>
</do>

 <p>
   <b>Order Information</b><br/>
   Address: <input type="text" name="Address"/>
   Credit Card: <input type="text" name="CreditCard" />
 </p>
</card>

</wml>
```

finishorder.jsp

LISTING 18.6 The `finishorder.jsp` File Completes the User's Order and Adds It to the Order Database Table

```
<% response.setContentType("text/vnd.wap.wml"); %>
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">

<%@ page import="java.sql.*, javax.servlet.http.Cookie" %>

<%
    Connection dbConn = null;

    // get a connection to the database
    try
    {
        // instantiate the db driver
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        // setup the connection
        dbConn = DriverManager.getConnection(
            "jdbc:odbc:shop"
            );

        // create a statement on the connection
        Statement userStmt = dbConn.createStatement();
```

LISTING 18.6 Continued

```
        // issue the SQL statements
        userStmt.executeUpdate(
        "INSERT INTO Orders(Username, Address, Phone, CreditCard,
➥OrderID) SELECT 'WEB' as EXPR1, '" + request.getParameter("Address")
➥+ "' AS EXPR2, '555-1212' as EXPR3, '" + request.getParameter("CreditCard")
➥+ "' as expr4, " + request.getParameter("CartID") + " as expr5"
        );

        userStmt.executeUpdate(
        "INSERT INTO Orders_Products(OrderID, ItemID, Quantity)
➥ SELECT ShoppingCartID, Item, Quantity FROM ShoppingCart_Items
➥WHERE ShoppingCartID=" + request.getParameter("CartID")
        );


    }
    catch(Exception e)
    {
    out.println("Exception!!!");
    out.println(e.getMessage());
    return;
    // assume no error conditions, for now
    }

%>

<wml>
    <card id="Notify" title="NotifyUser">
    <p>
            <b>Thanks for your order!</b><br/>
            <a href="login.jsp">Place a new order</a>
    </p>
   </card>
</wml>
```

Summary

E-commerce is a highly practical and exciting application for mobile users. For this fourth and final sample application, we showed you a complex example of how an e-commerce application might work using JSP and WAP. We showed you what kinds of data we needed to store in the data source, and how the set of tables we created were used throughout the application.

Next, we showed you how to keep track of the user's shopping session with a specific User ID that was passed throughout the application. Once that was established, we dynamically generated categories and product lists, so that the application would be scalable and able to adapt to changing products and additional categories. To keep track of the users' selections, we created a familiar "shopping cart," which held items until they were ready for purchase. Finally, we created a way for the user to purchase those items by collecting purchase information and storing that data with the product data in a separate table.

Although this e-commerce application was fairly straightforward, we built it in such a way that additional products or categories could, at any time, be added to the data source without requiring any changes to the code shown here. This kind of scalability will be critical for other kinds of e-commerce applications, because products and pricing can change at a moment's notice.

WAP is a powerful way to interface with mobile devices. When joined with a more dynamic technology, like JSP, the full potential of WAP can be realized in all kinds of applications, such as e-commerce. It is then that we can bring any kind of data to anyone, anywhere, using WAP.