SYNGRESS®

1 YEAR UPGRADE
BUYER PROTECTION PLAN

SYNGRESS.COM 1 YEAR UPGRADE BUYER PROTECTION PLAN

.NET Mobile Web Developer's Guide

Develop and Deliver Enterprise-Critical Mobile Web Applications

- Complete Case Studies with Ready-to-Run Source Code and Full Explanations
- Hundreds of Developing & Deploying, and Debugging Sidebars, Security Alerts, and .NET FAQs
- Complete Coverage of Web Services

Steve Milroy
Ken Cox
DotThatCom.com
Doug Safford
Laura Barker
Amit Kalani
Wei Meng Lee Technical Editor

SYNGRESS®

1 YEAR UPGRADE
BUYER PROTECTION PLAN

.NET Mobile
Web Developer's Guide

Steve Milroy
Ken Cox
DotThatCom.com
Doug Safford
Laura Barker
Amit Kalani
Wei Meng Lee Series Editor

KEY	**SERIAL NUMBER**
001	DJG4T97HG4
002	AKMSD8QE24
003	V98BF3N54N
004	ZT52S29U8N
005	8TR55U6N7H
006	NFG4RW23C4
007	BX6MK3TR46
008	GFUR565MER
009	83N5MBH8KW
010	GT6YVT32FC

PUBLISHED BY
Syngress Publishing, Inc.
800 Hingham Street
Rockland, MA 02370

.NET Mobile Web Developer's Guide

Printed in the United States of America

1 2 3 4 5 6 7 8 9 0

ISBN: 1-928994-56-3

Technical Editor: Wei Meng Lee
Technical Reviewer: Chris Lovett
Co-Publisher: Richard Kristof
Acquisitions Editor: Catherine B. Nolan
Developmental Editor: Kate Glennon

Project Editor: Maribeth Corona-Evans
Cover Designer: Michael Kavish
Page Layout and Art by: Shannon Tozier
Copy Editor: Darren Meiss
Indexer: Nara Wood

Distributed by Publishers Group West in the United States and Jaguar Book Group in Canada.

Acknowledgments

We would like to acknowledge the following people for their kindness and support in making this book possible.

Ralph Troupe, Rhonda St. John, and the team at Callisma for their invaluable insight into the challenges of designing, deploying and supporting world-class enterprise networks.

Karen Cross, Lance Tilford, Meaghan Cunningham, Kim Wylie, Harry Kirchner, Kevin Votel, Kent Anderson, and Frida Yara of Publishers Group West for sharing their incredible marketing experience and expertise.

Mary Ging, Caroline Hird, Simon Beale, Caroline Wheeler, Victoria Fuller, Jonathan Bunkell, and Klaus Beran of Harcourt International for making certain that our vision remains worldwide in scope.

Annabel Dent of Harcourt Australia for all her help.

David Buckland, Wendi Wong, Marie Chieng, Lucy Chong, Leslie Lim, Audrey Gan, and Joseph Chan of Transquest Publishers for the enthusiasm with which they receive our books.

Kwon Sung June at Acorn Publishing for his support.

Ethan Atkin at Cranbury International for his help in expanding the Syngress program.

Jackie Gross, Gayle Vocey, Alexia Penny, Anik Robitaille, Craig Siddall, Darlene Morrow, Iolanda Miller, Jane Mackay, and Marie Skelly at Jackie Gross & Associates for all their help and enthusiasm representing our product in Canada.

Lois Fraser, Connie McMenemy, and the rest of the great folks at Jaguar Book Group for their help with distribution of Syngress books in Canada.

Contributors

Doug Safford (MCSD) is a Senior Architect/Project Manager with Empowered Software Solutions (ESS) in Burr Ridge, IL. ESS is a Chicago-based consulting firm and Microsoft Gold Certified Partner in e-Commerce. Doug has over 11 years experience in application development. His current focus is on leveraging the .NET Framework for design and development of large enterprise applications. His typical assignment consists of assisting large clients in enterprise application design, then mentoring the clients' architects and developers on how best to implement their application designs. He is a frequent speaker at .NET and corporate user group meetings. Doug would like to thank his wife Cindy for all her support. And to Justin and Melanie, you are my happy thoughts. Also thank you to the folks at ESS for providing support.

Laura Barker (MCSD) is a Project Manager and Senior Developer/Analyst for Empowered Software Solutions (ESS), a Chicago-based consulting firm and Microsoft Gold Certified Partner in e-Commerce. Laura recently led the project management of an Extranet insurance and financial site, written completely with Microsoft .NET tools and technologies. The site utilized ASP.NET, VB.NET, SQL Server 2000, ADO.NET, BizTalk Server, and COM+ Services. She is a regular presenter for local area businesses and at the Chicago .NET User Group, on various topics of .NET and will be speaking at VSLive! 2002 San Francisco on SQL Server 2000 and XML. Laura has authored the article, "Expand Your Business with Mobile Applications," for *.NET Magazine*. She has over 12 years experience in Information Technology and has experienced all aspects of life cycle development. She has acted as project manager, developer, or analyst on several enterprise wide and Web solutions all focused on Microsoft technologies. Laura would like to thank her family for their love and support.

Amit Kalani (MCP), contributing author of *Inside ASP.NET,* is a Senior Consultant with CIStems. Amit programs extensively using Microsoft .NET Framework, with special focus on C# and the CLR. He has done technical editing for several popular books on subjects including C#, VB.NET, ASP.NET, and SOAP. Amit has also been involved in training where he has designed and delivered courses on subjects like C, Data Structures, Systems Analysis and Design, ASP, and Visual InterDev. He has a bachelor of science degree from the University of Rajasthan and also has DOEACC Level A certification. Amit currently resides in Wixom, MI with his wife Priti.

Steve Milroy is a Mobile/Wireless Technologist and is currently with a nationwide e-solutions company, Immedient. He is working on a number of WAP, VoiceXML, and wireless middleware projects. Steve has presented at developer conferences, user groups, and business associations on various wireless development and infrastructure topics. Originally from Sydney, Australia, Steve now resides in the United States. This experience in the Australasian and US markets gives him a broad and unique view of wireless around the world.

Norman Gragasin (MCSD) is a Senior Solution Developer Architect for Empowered Software Solutions, Inc. (ESS). ESS is a Chicago-based consulting firm and a Microsoft Gold Certified Partner in e-Commerce. Norman has over 13 years of experience in analysis, software architecture, development, project leadership/management, and implementation of software systems. He is one of the Directors of the Chicago.NET Users Group and a presenter on such topics as ADO.NET. The Chicago.NET Users Group focuses on the Microsoft .NET Framework and educating developers on the tools and technologies of the Microsoft .NET Framework. Norman has also given technical presentations at the Visual Basic Developer's Network on Microsoft Visual Interdev and ActiveX Data Objects and OLEDB. He has authored a feature article, "Integrate Apps with BizTalk", in *Visual Studios Magazine,* where he describes how developers can use BizTalk Server and Microsoft .NET tools and technologies to integrate applications to provide business solutions. Norman would like his family for their love and support.

Ken Cox is a Technical Writer and Web Applications Developer in Toronto, Canada. He is a frequent contributor to computer books including *Inside ASP.NET* and *Teach Yourself Object Oriented Programming in VB.NET in 21 Days*. Ken also writes articles and software reviews for publications that include *Visual Studio Magazine* and *Web Techniques*. During his six years at Nortel Networks in Toronto, Ken was a Senior Technical Designer in the documentation department, writing and developing leading edge forms of user assistance such as interactive Web-based tutorials. His writing has won several awards in technical writing competitions. Before launching into his second career as a technical writer, Ken spent 20 years as a broadcast journalist in Toronto and Quebec City for Canada's top radio stations and news networks. A technophile, Ken was a member of the original beta test group for Active Server Pages (ASP) when it was still known by its code name, Denali. Microsoft has honored Ken as a Most Valuable Professional (MVP) in recognition of his expertise in Internet technologies and volunteer contributions to the user community. He holds a bachelor's degree in Radio and Television Arts from Ryerson University and is a senior member of the Society for Technical Communication. Ken currently lives in Toronto with his wife Vilia.

David Jorgensen (MCP) is an Instructor at North Seattle Community College, University of Washington Extension campus, and Puget Sound Centers. He is also developing courses for Seattle Vocational Institute, which teach .NET and Web development to the underprivileged in the Seattle area. David also provides internship opportunities through his company DotThatCom.com, which does online sample classes and chapters of books. David holds a bachelor's degree in Computer Science from St. Martin's College and resides in Puyallup, WA with his wife Lisa and their two sons Scott and Jacob. David is a contributor to Syngress Publishing's *ASP.NET Web Developer's Guide* (ISBN: 1-928994-51-2) and *C# .NET Web Developer's Guide* (ISBN: 1-928994-50-4).

Patrick Coelho (MCP) is an Instructor at The University of Washington Extension, North Seattle Community College, Puget Sound Center, and Seattle Vocational Institute, where he teaches courses in Web Development (DHTML, ASP, XML, XSLT, C#, and ASP.NET). Patrick is a Co-Founder of DotThatCom.com, a company that provides consulting, online development resources, and internships for students. He is currently working on a .NET solution with contributing author David Jorgensen and nLogix. Patrick holds a bachelor of science degree from the University of Washington, Bothell. He lives in Puyallup, WA with his wife Angela. Patrick is a contributor to Syngress Publishing's *ASP.NET Web Developer's Guide* (ISBN: 1-928994-51-2) and *C# .NET Web Developer's Guide* (ISBN: 1-928994-50-4).

Technical Reviewer

Chris Lovett is an Architect on the XML team at Microsoft Corporation where he has been involved in the development of XML technologies since early 1997. He worked on the Microsoft XML Parser in Java reference release, the MSXML COM component, which is used in Internet Explorer, Windows 2000, SQL 2000, and Exchange 2000. He also designed the System.Xml classes in the .NET Frameworks. Chris also writes articles and whitepapers for the Microsoft Developer Network. He has more than 10 years experience in networking and operating systems development working at companies such as IBM and Taligent as well as other Silicon Valley startups. Chris received a bachelor of science and mathematics degree from the University of New South Wales in Sydney, Australia.

Technical Editor and Series Editor

Wei Meng Lee is Series Editor for Syngress Publishing's .NET Developer Series. He is currently lecturing at The Center for Computer Studies, Ngee Ann Polytechnic, Singapore. Wei Meng is actively involved in Web development work and conducts training for Web developers and Visual Basic programmers. He has co-authored two books on WAP. He holds a bachelor's degree in Information Systems and Computer Science from the National University of Singapore. Several books in the .NET series, *VB.NET Developer's Guide* (ISBN: 1-928994-48-2), and *ASP.NET Developer's Guide* (ISBN: 1-928994-51-2) and *C# .NET Developer's Guide* (ISBN: 1-928994-50-4) are currently available from Syngress Publishing.

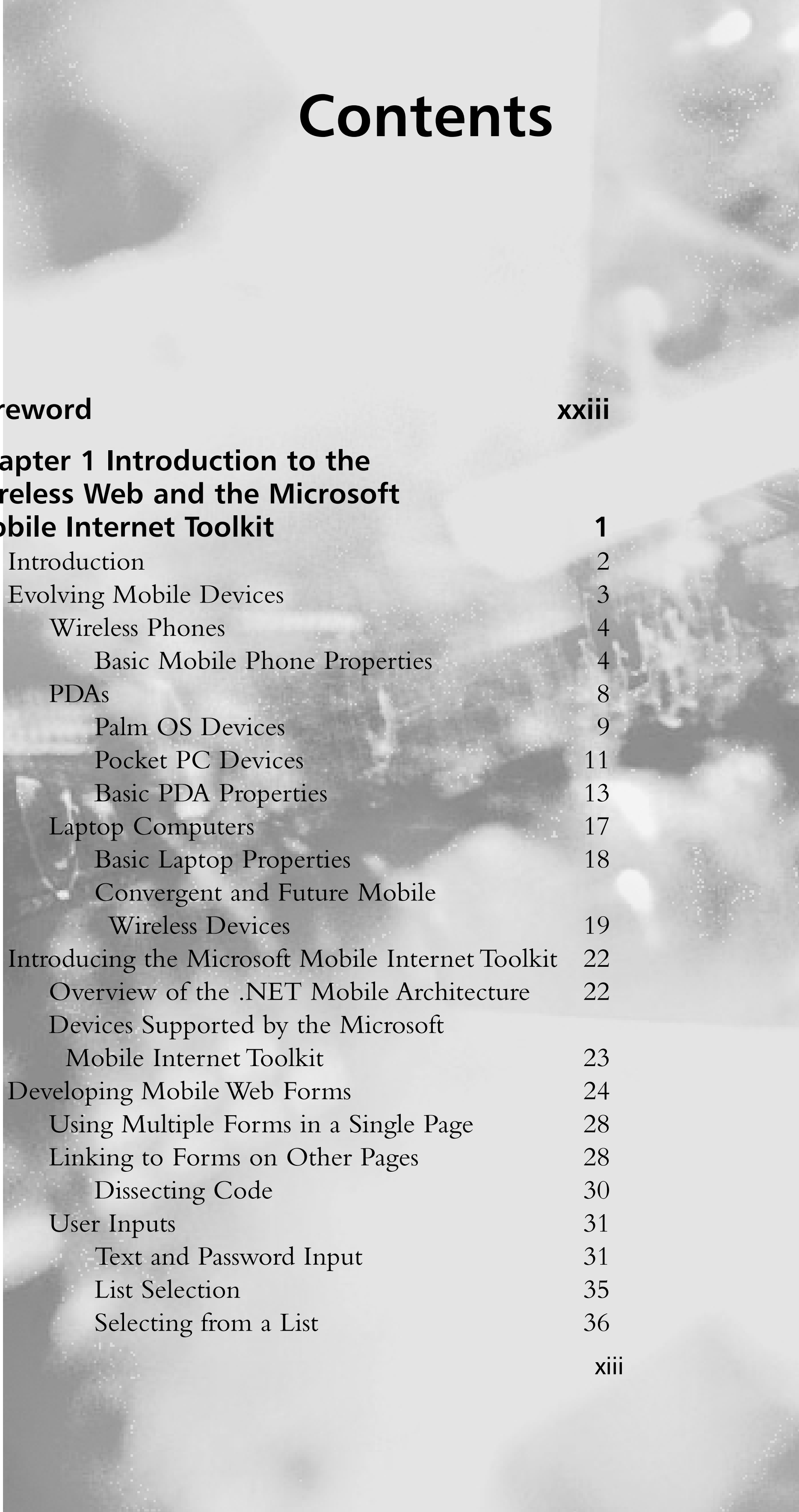
About the Web Site

The Syngress Solutions Web Site contains the code files that are used in specific chapters of this book. The code files for each chapter are located in a "chXX" directory. For example, the files for Chapter 6 are in ch06. Any further directory structure depends on the projects that are presented within the chapter.

Chapters 6, 7, and 8 contain code that apply to the situations described in their sections. This code will be extremely useful for understanding and enhancing the way you use the Microsoft Mobile Internet Toolkit. Specifically, Chapter 6 deals with accessing data using ADO.NET. Chapter 7 takes you from start to finish with a mobile solution using key Microsoft technologies. And lastly, Chapter 8 builds a mobile movie ticket purchasing application which will allow users to purchase theater tickets using mobile devices such as Web enabled cell phones and PDAs.

SYNGRESS syngress.com

Look for this icon to locate the code files that will be included on our Web site.



Contents

Understanding Basic Mobile Phone Properties

The primary purpose of mobile phones is to enable the original *killer* app: voice communication. As such, they need to have minimal requirements for memory and processing power:

- **Connectivity** 9.6 Kbps digital cellular
- **Screen size** Typically 3 x 2.5 cm (1.25 x 1 in.) equivalent to 5 lines of text
- **Memory** Minimal
- **Processing power** Minimal

Using the Compilers

The compilers that ship with the SDK are csc.exe (C#), vbc.exe (VB.NET), and jsc.exe (Jscript .NET). The command *vbc /t:exe /verbose Module1.vb* translates to the following:

- *vbc* = Run the vbc.exe compiler with the following options.
- */t:exe* = The "target" for the compilation is an exe file.
- */verbose* = Display all of the output information.
- *Module1.vb* = The source file that we want to compile.

Answers to Your Frequently Asked Questions

Q: Can a page have a mix of .NET languages within it?

A: No, the page and the project it resides in should all be created within the same .NET language. However, referenced components used within the project can be of any .NET language. This is because all code gets compiled down to the same Microsoft Intermediate Language (MSIL). At this level, the code is all the same language.

Developing & Deploying…

Emulator versus Real Device

Emulator programs are your best friends for wireless application development because they provide productive environments with powerful tools for developing, testing, and debugging. Also, it is much more economical to install multiple simulators on a developer's machine rather than purchasing real devices for each of the developers in a development team.

Using Mobile Web Forms

Mobile Web Forms are the foundation of developing a mobile Internet application with the Mobile Internet Toolkit, and they significantly simplify the development of mobile Internet applications.

Exploring End-to-End Microsoft Mobile Solutions

Microsoft Mobile Information Server (MIS) 2001 helps you tie together mobile applications and gives your Wireless Application Protocol (WAP) and Pocket PC users access to services on your intranet. By tying MIS with Exchange Server 2000, you open a rich array of desktop services to these tiny portable devices.

Debugging…

Some Mobile Devices Do Not Support Cookies

If one or more of the mobile devices you need to support do not support cookies, your ASP.NET pages may not function correctly. You may get errors stating that your session has expired or that the client did not send a valid cookie method. You can easily fix this problem:

1. Open Web.config.
2. Find the element *sessionState*.
3. Set the *sessionState* attribute cookieless = "*true*".
4. Save and rebuild your project.

Foreword

Just a few years ago, Web developers were struggling to keep up with the features of the two main Web browsers in the market—Microsoft Internet Explorer and Netscape Navigator. We were all victims of the Web browser war between Netscape and Microsoft. With each browser supporting a different set of features, a lot of time and effort was spent customizing Web pages so that they looked their best on each browser. Just when we thought that life could not get any more complicated, a new breed of devices suddenly appeared on the market. Anyone who has spent a significant amount of time developing for mobile devices can attest to the fact that developing for mobile devices is a totally different endeavor compared to Web development.

One of the challenges that mobile application developers have to face is the different look and feel of an application on different devices. An application that runs perfectly on one device may be totally unusable on another. As a result, developers often have to learn new languages such as the Wireless Markup Language (WML), Handheld Device Markup Language (HDML), and Compact HTML (cHTML). One solution has been for developers to either create multiple sets of documents or use the complex XML Stylesheet Language Transformation (XSLT) to tailor their content for multiple devices. But using XSLT is not an easy solution and requires an extensive programming foundation.

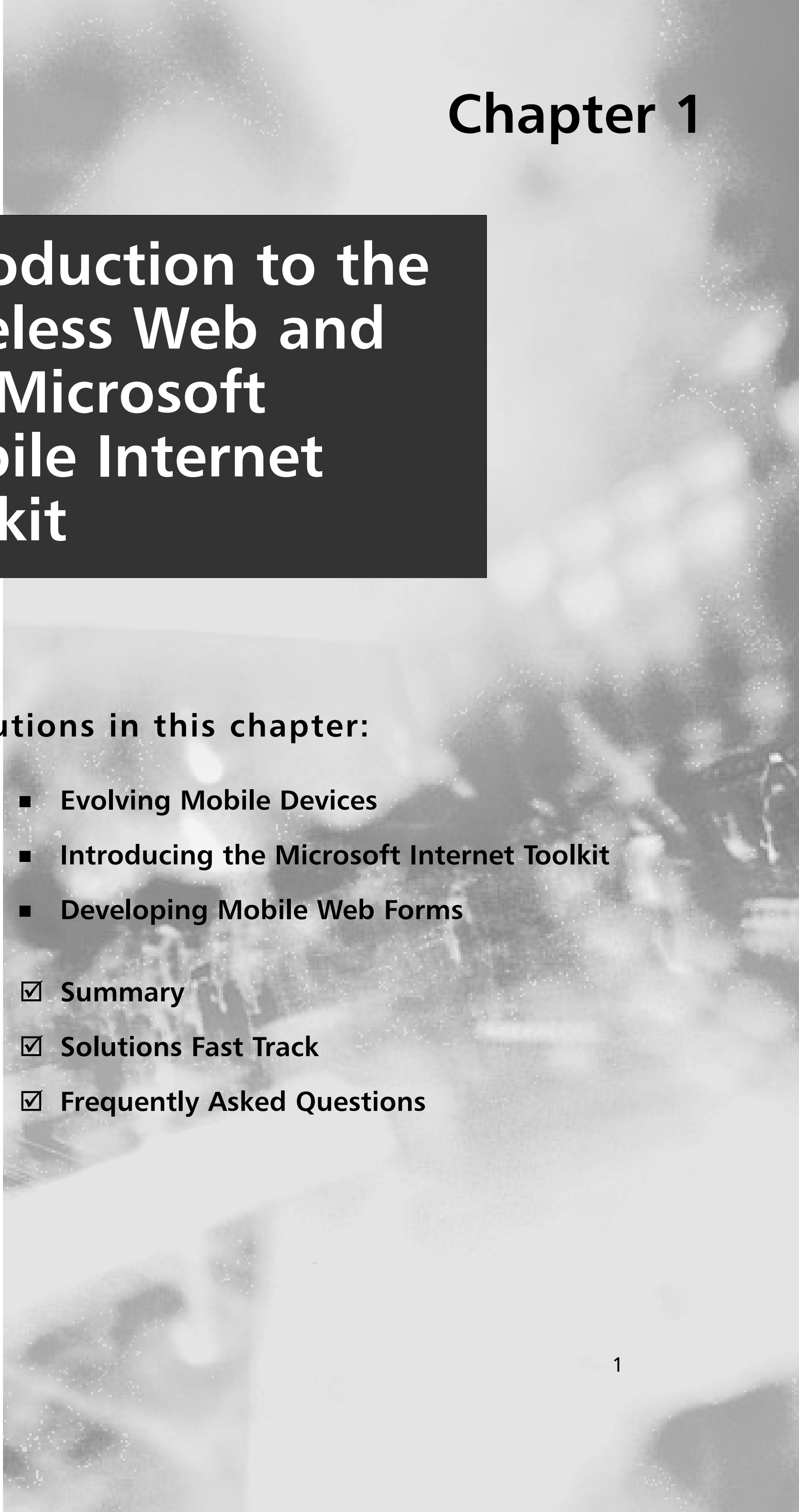
To solve this customization problem, several solutions have been proposed. The most popular one is to write your application in a proprietary language. During runtime, an application server translates your application into the desired output language. While such a solution is feasible, developers often have to learn additional languages, thereby increasing the cost and duration of projects.

Together with the introduction of the .NET Framework, Microsoft announced the Microsoft Mobile Internet Toolkit (formerly known as the .NET Mobile Web SDK). With the Microsoft Mobile Internet Toolkit, developers can now leverage their core strengths in Web development using technologies such as ASP.NET and ADO.NET to develop compelling mobile applications. There is now no need to customize the output of your application anymore; the .NET Mobile Framework will take care of it, leaving developers to focus on the business logic of the application. Even better, the toolkit is integrated with Visual Studio.NET, giving developers a familiar platform to develop mobile applications.

The *.NET Mobile Web Developer's Guide* will provide readers with a complete guide to developing mobile applications using Microsoft technologies. We focus on using ASP.NET and the .NET mobile SDK to provide an introduction to the .NET platform. In addition, *.NET Mobile Web Developer's Guide* will give you the insight to use the various Microsoft technologies for developing mobile applications.

We are all very excited with the opportunities opened up by the Microsoft .NET mobile framework. Come on in, and see how to create killer apps with this great new tool!

Wei Meng Lee, Series Editor
Syngress .NET Developer Series



Chapter 1

Introduction to the Wireless Web and the Microsoft Mobile Internet Toolkit

Solutions in this chapter:

- Evolving Mobile Devices
- Introducing the Microsoft Internet Toolkit
- Developing Mobile Web Forms

☑ Summary

☑ Solutions Fast Track

☑ Frequently Asked Questions

Introduction

The past century has brought about many changes in information and communications technology, from the invention of the telephone and broadcast technologies to the invention of the personal computer and the Internet. These changes have enabled us to exchange information with other individuals and to retrieve data from vast databases practically instantly. The wireless Internet is a new revolution upon us, one that will affect the world on a scale similar to that of the wired Internet. We have seen it grow in Europe and Asia, and North America appears to be the next frontier of this expansion.

We now live in a world populated with various devices that are capable of exchanging information at unprecedented rates of speed, measured on the scale of milliseconds. We have mobile telephones, pagers, personal digital assistants (PDAs), and laptop computers, all capable of being connected to the Internet. It is truly an exciting time to be alive.

In this chapter, we provide a brief overview of wireless technology, discussing some of the devices that are currently connectable. We also cover in brief some of the similarities and differences between the wired and wireless Internet. We briefly discuss the concept of mobile versus fixed wireless and provide some examples of these different types of wireless connectivity in action.

We then begin to examine the Microsoft Mobile Internet Toolkit and how it can aid in mobile application development. The concepts and application development techniques presented in this chapter will be covered in greater detail throughout the book. The Microsoft Mobile Internet Toolkit is a set of mobile framework extensions that have been added to ASP.NET Web Forms. With these extensions, a mobile application developer can create compelling mobile applications without worrying unduly about the limitations of the various target devices. The current situation in mobile application development is that various devices have a very different look and feel, and often developers have to spend huge amounts of time tailoring their applications to run on the target devices. A typical solution is to code the content of your application in XML and use XSLT to transform the content into a target markup language like Wireless Markup Language (WML).

Rather than focusing on the user interface issues, the Microsoft Mobile Internet Toolkit provides a set of APIs to let the developer concentrate on the functionality of the application. During runtime, the Microsoft Mobile Internet Toolkit API will automatically detect the kind of device accessing the application and generate the appropriate codes to run on it. To get the full benefit from this chapter, you should know the basics of Microsoft ASP for developing Web applications.

Evolving Mobile Devices

The mobile landscape today is in a state of continual change. We hear of new devices introduced to the market almost weekly, and wireless access options continue to multiply. Although detecting the exact device accessing your server is possible in most cases, the sheer variety of different devices means that you probably won't want to format content for each one. The good news is that most of the devices likely to be accessing your site wirelessly fall into three broad categories—mobile phones, PDAs, or laptop computers. Each has its own unique advantages and disadvantages. Although significant differences exist between devices in each category—PDAs in particular come in a wide variety of configurations—the three main categories are differentiated by connectivity, screen size, memory, and processing power.

The most widely available wireless devices are mobile phones. Their primary purpose, of course, is voice communication. With the addition of data services from the wireless carrier, they also work well for short text messages (using Short Message Service [SMS]) and sometimes for reading e-mails, but the difficulty of entering text makes them cumbersome for sending e-mail. Wireless Application Protocol (WAP) phones also allow you to access specially formatted Internet pages.

Traveling professionals have been using PDAs for several years now to track schedules, store contact information, and enter expenses while on the road. With the addition of a wireless connection, their usefulness is increased. With larger screens and handwriting recognition interfaces, they are suitable for short e-mails and can also be used to access the Internet.

Laptops have always been mobile, of course. Laptops with a wireless modem in the PC Card slot eliminate the need to search for phone jacks, fiddle with wires and connectors, or huddle in public phone booths. One advantage laptops, and some PDAs, have over wireless phones is that the wireless component is upgradeable, so that as better, faster options become available, users don't need to discard the whole device. With the current pace of development in the wireless Web, this is probably a sensible precaution, if you have the option.

Several other devices are available that seek to combine aspects of each category—a mobile phone with an integrated Palm screen, PDAs that can be used as phones, and laptop-sized devices without keyboards that you use by writing directly on the screen.

Wireless Phones

The first and still most prevalent device today is the data-enabled cellular phone. Almost all of the major cellular carriers now offer data services as well as the traditional voice service. All of the major handset manufacturers—Nokia, Motorola, Ericsson, Mitsubishi, Alcatel, and others—offer data-capable phones, and before long, this will be standard on all new phones. These are typically the same size as regular cell phones, but with a screen capable of displaying specially formatted text. They use the WAP protocol. WAP was developed as an alternative to Hypertext Transfer Protocol (HTTP) to deal specifically with the restrictions of the current generation of wireless, that is, with low speeds and high latency. For display on WAP phones, content needs to be coded in WML. WAP phones don't connect directly with WML Web servers. They communicate with special WAP gateways, typically operated by the carriers, which then forward the request to the content server on their behalf. The WML content returned is then compiled into a special compressed format before being sent back to the WAP phone, where an application called a microbrowser decodes and displays it.

Basic Mobile Phone Properties

Mobile phones are, first and foremost, phones. Their primary purpose is to enable the original *killer app*: voice communication. As such, they need to be small and light and have minimal requirements for memory and processing power:

- **Connectivity** 9.6 Kbps digital cellular
- **Screen size** Typically 3 x 2.5 cm (1.25 x 1 in.) equivalent to 5 lines of text, about 15 characters per line
- **Memory** Minimal
- **Processing power** Minimal

Mobile Phone Connectivity

A data-enabled mobile phone uses the same radio-frequency (RF) connection as your voice calls to connect with its base station. This is typically a cell tower somewhere within a few miles. Although it depends on a number of factors, such as distance from the cell tower and number of users within that cell, the rated data speed in most cases is 9.6 Kbps (some services offer 14.4 Kbps). Compared to a 56 Kbps dial-up modem, the minimum connection speed most Web sites are designed for, you can see this is quite slow. In addition to low bandwidth, the current cellular networks suffer from high latency—that is, a significant delay occurs between the time a user hits a Submit button and when the resulting content is sent back to the device. It's also not uncommon for the signal to be dropped in the middle of a transaction as the user drives into a tunnel or the radio shadow of a large building.

The signal between the handset and the base station is encrypted and compressed. From there, the signal is routed over regular landlines to a special server called a WAP gateway. The segment of the call from the handset to the gateway is done using Wireless Session Protocol (WSP), a protocol defined within WAP. The WAP Gateway then acts on the phone's behalf to request the page from your server using traditional HTTP. The concept of the WAP gateway may be unfamiliar to you if you're accustomed to the traditional Internet client/server model. The gateway is basically acting as an agent or proxy for the wireless device and also translates from the WAP protocol stack to the TCP/IP stack used on the Internet. This is quite important to remember: A mobile phone never communicates directly with your Web server; it is always a WAP gateway acting on its behalf. Because the gateway can have a significant effect on how your content is displayed, looking at this a little more closely is worthwhile.

When a user requests some content (either by typing a URL directly into the phone's microbrowser or by clicking on a link), the following series of steps occurs:

1. The handset establishes a connection with its base station.
2. Once this connection is set up, the microbrowser then initiates a connection to a WAP gateway predefined in the phone's configuration.
3. The microbrowser requests a URL from the WAP gateway. This is done via a compact binary encoded request.
4. The gateway translates this request into an HTTP request and sends it over the wired Internet to the specified content server.
5. The content server responds by sending a page of WML content, which may also contain WMLScript (similar to JavaScript) and special graphics in WBMP format.
6. The gateway compresses the response into a special binary format optimized for low-bandwidth networks, then sends it back to the microbrowser. It also compiles any WMLScript found in the response.
7. The microbrowser decodes the compressed signal, and attempts to display it, if possible.

As you can see, quite a few steps take place between a visitor and your content, and each of the components along the way can have a significant effect on the format of your content. You need to understand the effect each can have on the data you send to your visitors. To add to this, the same components but by different manufacturers can behave quite differently. This is analogous to the early days of the Web, when you had to contend with different manufacturers' browsers displaying your HTML in different ways. A WAP phone contains a microbrowser, which is similar in function to the familiar desktop browser. However, several major microbrowsers are in circulation, and though each conforms to the WAP specification, the specification allows for quite a lot of flexibility in how they actually display content.

The gateway, which is typically housed at the cellular carrier's premises, may also alter the content somewhat on its way through. Some gateways, for instance, store and pass cookies, whereas some do not. The gateway can also add special header fields, and it sometimes removes header information. The gateway will also cache information on behalf of the phone, because most phones don't have enough local memory to save much data. Again, this varies from one gateway to another, so you generally can't rely on it.

Mobile Phone Screen Size

The size and resolution of the display screen is probably the biggest hurdle you'll face in developing Web sites for WAP phones. This is similar to the early days of the Web, when you could never be sure of the screen resolution or color capability of visitors' monitors. Some phones have a mechanism whereby they can send capabilities information—such as pixel count, number of lines of text, and number of soft keys—to your server. Unfortunately, not all phones provide this information, and not all gateways pass it on.

A typical phone screen is 3 x 2.5 cm (1.25 x 1 in.) and usually has a monochrome LCD capable of displaying only black or white. Most current phone screens are limited to displaying about 5 lines of text, with about 15 characters per line. A few models have slightly larger screens, and some are even smaller. It is possible to detect the incoming User Agent (the microbrowser in the phone), compare this to a database of known phone configurations, and then format your content accordingly, but the sheer variety of possible handset configurations makes formatting your content for specific models of phone very problematic. Most people will choose a lowest common denominator format that has been tested to work satisfactorily on most common phones.

The minimal screens mean that you'll need to rethink the amount of content you put on pages meant for WAP users. People can always scroll up and down, of course, but reading in this manner is difficult. Long text pieces simply don't work in this form, so you'll need to cut down drastically on the amount of text on your pages. Fitting navigation menus on there as well becomes a difficult task. WML actually contains some features to help in this regard. Because most phones have a number of *soft keys* (buttons below the screen to which you can assign menu items), you can shift some of the navigational elements, such as Home, Back, and Next, off the main screen. However, the utility of this feature is reduced significantly because each manufacturer implements these soft keys in very different ways, both physically and logically. Because you won't know exactly how the buttons will implement your interface on all phones, designing interfaces becomes something of a guessing game.

Mobile Phone Memory

Most wireless handsets have little or no memory available for storage. They do have some storage for personal phone numbers, but this varies from phone to phone, which means that you have to be very careful how much data you send to a handset at one time. Gateways compress your WML before sending to the device, but how much compression happens varies by gateway. In particular, because you typically won't know how much data the phone can handle, you'll need to pick a safe limit you're sure will work on most phones. Because gauging how much compression different carrier gateways will provide is difficult, this may take some trial and error, but as a general rule you should keep your pages, or WML decks, under 1.5 Kb total. This may mean developing special server code if you're doing things such as returning database record sets; you'll need a way to measure the size of the record set returned by a query and then split it up into WAP-sized chunks.

However, WML does allow for something that generally doesn't exist on the Web: persistent client-side variables. This means that you can capture form entries on one page and then pass them to another page without requiring a trip back to the server. You could, for instance, ask a visitor for some input on one card of a multicard deck and use their responses to determine which card to navigate them to next. This kind of conditional branching is very difficult to achieve via HTML alone. Another potential use might be to store a visitor's answers to a question from one page, then refer back to these answers several pages later, without needing to transfer the data back to the server and store it there. Again, these variables are limited by available handset memory, but they are session-independent, meaning they will be stored on the handset, even after your visitor navigates away from your site. However, as new data arrives, these variables may be pushed out and replaced. Furthermore, it is possible for any site to clear *all* of the variables on the phone.

SECURITY ALERT!

Unlike cookies on the Web, which can be accessed only from the same domain that set them, WML client variables are available to any Web site as long as they remain in memory. So if, for instance, you were to set a variable and value "password=abc123", you raise the potential for a malicious WAP site to access and save this.

Mobile Phone Processing Power

The current crop of mobile phones has minimal processing power—basically just enough to run an embedded operating system, and a few simplistic games. Bear this in mind if you've got very complicated WMLScript that you expect to be processed on the device. Heavy-duty computation tasks are better handed back to the server to process. Higher powered phones capable of downloading and running Java programs are beginning to appear on the market, particularly in Japan, but these are so far not widely available in the U.S.

PDAs

The next step up in device size is the PDA. These come in many different forms, but typically have a larger screen, more memory, and more processing power than mobile phones. A PDA generally refers to a device small enough to hold in the palm of the hand, but with a larger screen than the typical mobile phone. Current PDAs evolved from gadgets designed to help you manage your contacts and calendar—essentially electronic FiloFaxes—and were relative latecomers to the wireless Internet. The market for PDAs is split mainly between those running the Palm operating system from both Palm, Inc. and its licensees (Handspring, IBM, Sony, and Symbol), and devices based on Microsoft's Windows CE, with a couple of niches occupied by other alternatives such as Symbian's EPOC and other devices.

One thing to bear in mind with PDAs is that, even if the units are company-supplied, these are fundamentally personal devices. People carry these with them constantly, and use them to track personal schedules, birthdays, grocery lists, and address books, just as much as they do company work. Businesses have been slow to adopt these devices, although that is now beginning to change. In fact, these devices first began to enter corporations when people brought their own devices to work and began synching up with their corporate calendars and address books.

Developing & Deploying…

Blackberry: Pager or PDA?

A device that has become quite popular, particularly with corporate "road warriors," is the RIM 957—popularly known as the Blackberry—from the Canadian firm Research in Motion. This pager-like device features a miniscule keyboard and an always-on connection to corporate e-mail systems, such as Microsoft Exchange. The first version of this device had a small three-line screen, but the RIM 957 added a screen with the same dimensions and resolution as the Palm. Corporate users in the U.S. have found this device almost addictive. Utilizing North American CDPD networks, the device constantly polls a dedicated Blackberry server connected to the corporate mail server for new e-mails and downloads them automatically, giving the impression of always-on, anytime, anywhere e-mail access. First rolled out in North America, the Blackberry was such a success that it is now being made available in Great Britain in partnership with British Telecom, utilizing their GPRS service.

Palm OS Devices

Although there were earlier attempts, Palm, Inc.'s device was the first commercially successful PDA. When it was introduced in 1996, the Palm Pilot was an instant success due to its ease of use, intuitive user interface, and small size. Although the casings have changed quite a bit since then, and more memory has been added, the actual Palm operating system has changed very little over the years. A large community of developers has grown up around it, so a huge variety of programs are now available. Until quite recently, Palm, Inc.'s primary market was individual users. Even though Palm device users tend to be extremely loyal, Palm, Inc. has realized that to maintain their market position they need to develop enterprise-level applications and market to large corporations. To make their PDA acceptable to corporate IT managers, they also need to address concerns of security and support, and they need to beef up its meager memory and processing power to make it capable of running enterprise-class applications.

Palm, Inc. also licenses its OS to several vendors. Handspring, founded by the original developers of the Palm OS, took a leaf from Apple Computer's book and released a series of very stylish devices in the Visor line. Although the basic OS remains almost the same, Handspring sells Visors with a variety of colorful translucent cases and developed a unique, proprietary expansion slot called the SpringBoard, which allows other manufacturers to make add-on modules for functions such as wireless access, Global Positioning System (GPS), and even a module that turns the Visor into a mobile phone. Sony's Clié adds a special jog-wheel that allows for improved navigation around the screen, and also has a model with a higher screen resolution. IBM rebrands the Palm OS as its WorkPad line, which it markets to corporations. Symbol and a few other companies take the basic Palm device and encase it in a rugged, weather-resistant housing, adding an integrated barcode scanner and wireless LAN access to make units for use in warehouse management and other industrial applications.

Palm OS–based PDAs access the Internet via either a built-in modem (in the case of the Palm VII), or by means of a clip-on external modem, such as the one available for the Palm V from OmniSky. In the U.S., these modems typically use the packet-switched CDPD network mentioned earlier, whereas in Europe they use the GSM cellular standard. Most Palm devices currently on the market use low-resolution monochrome LCD screens, although Palm, Inc. and a number of its licensees have recently released some color models.

Palm, Inc.'s designers felt that the best solution to the limited screen size, and the very slow data speeds of wireless, was to do away with the concept of browsing as we understand it on the wired Web. Instead, they envisioned a way to give people quick access to targeted information, stripped of all embellishments. Palm, Inc. refers to this as *Web Clipping*. Rather than connecting directly with your content server, Palm devices generally use an intermediary server called a proxy. This is similar in concept to the WAP gateway, but it has quite different capabilities. Web Clipping uses a subset of HTML 3.2 with a few notable changes: It doesn't support frames, nested tables, or a lot of the formatting options of regular HTML.

The Palm.Net proxy reads Web pages on behalf of the device, and then it compresses them before sending them over the air. It will also rewrite any HTML that doesn't conform to the specification, including removing graphics wider than the Palm device screen size. However, the results of this translation are seldom what you had in mind. In most cases, you'll need to either construct new pages specifically for Palm OS, or reformat your existing pages so they work on both formats.

Developing & Deploying…

How Can I Validate the HTML in My Web Clipping Application?

The full Document Type Definition (DTD), which describes in detail the acceptable tags and attributes, is available at www.palm.com/dev/web-clipping-html-dtd-11.dtd. You can specify this DTD in your document and validate your code using the W3C validator at http://validator.w3c.org.

Visitors using the Palm OS generally won't type a URL into a conventional browser to access your site. Generally, they'll download a special Web Clipping Application (WCA, also previously referred to as a Palm Query Application [PQA]) from your regular Web site, then install this on their Palm device. WCAs are simply HTML pages compiled into a special binary format using an application that developers can obtain freely from Palm, Inc. One potentially useful feature of Web Clipping that differs substantially from traditional Web authoring is the ability to precompile graphics into the WCA, then later refer to these graphics from your online pages. Because the graphics are already resident on the Palm device, there is no need to download them over a slow wireless connection, which enables you to create extremely efficient applications. The drawback to this approach is that, because Web Clipping is a proprietary technology, you then can't use the same HTML for Pocket PC devices, which at present don't support this feature.

Pocket PC Devices

Pocket PC–based PDAs are a more recent addition to the mobile device arena, but they are gaining popularity because of their relatively higher-resolution color screens and greater processing power. Microsoft Pocket PC is a special version of Windows designed specifically for smaller devices, and it comes with familiar applications such as Outlook and Internet Explorer. In contrast to WAP phones and Palms, these devices generally make a direct HTTP connection with your server, without any intervening proxies.

After a few false starts with earlier versions, Microsoft's Pocket PC 3.0 revolutionized the PDA market when it was introduced in 2000. Although its market share is still considerably less than Palm's, it has raised the bar on functionality and continues to advance the state of the art in mobile wireless devices. The first and most obvious attribute is a higher resolution color screen (grayscale models are available, but these are largely confined to industrial units). Most models also have a backlit screen, making the display extremely bright and crisp. The Pocket PC operating system includes pocket versions of popular Microsoft applications, such as Word and Excel. It also has a version of Outlook that, with Microsoft ActiveSync, allows mobile users to sync the unit with their desktop or laptop Outlook. Most significantly for the wireless Webmaster, it features a browser that's very similar to Internet Explorer 3.2.

Rather than manufacture devices itself, Microsoft licenses its OS to any manufacturer that can meet the minimum technical requirements. These include a screen with 240x320 pixels of resolution, and memory of at least 16MB. 32MB is more common, and Compaq's iPAQ 3670 comes with 64MB. Pocket PC devices typically also have a more powerful CPU, allowing for more on-board processing.

One feature of Pocket PCs that's especially relevant to wireless is that most come with an industry-standard expansion slot; either CF or PCMCIA Type II (the same PC Card slot found on virtually every laptop computer). This immediately gives these devices a huge base of possible expansion options. When Compaq introduced their wildly popular iPAQ Pocket PC in 2000, other companies were quick to produce wireless options for the device, either writing software drivers for existing PCMCIA cards, or in some cases developing completely new PC cards.

Although technical features are obviously important, style has proven to be just as much of a selling point. When Compaq introduced their iPAQ, it was in such demand that units were back-ordered for months; it was practically impossible to get one through regular retail channels. At one point, iPAQs were selling on eBay for well over their retail value. Hewlett Packard makes a Pocket PC model, the Jornada 548, that's functionally very similar, but sales have slumped compared to the Compaq's superior visual appeal. At tradeshows and technology demonstrations throughout 2000 and 2001, the sleek and shiny iPAQ was *the* cool device to have.

Pocket PC–based PDAs have found ready acceptance too in the industrial market. Symbol, Intermec, iTronix and others make more rugged models based on the OS, usually with integrated barcode scanners and wireless connection options. The increased memory and more powerful CPUs make these devices suitable for applications that require more processing power on the handheld, such as mobile field service automation and sales force automation.

There is another class of mobile PDA device, sometimes referred to as Handheld PC or *clamshell* form factor. These are devices that feature a horizontal display aspect ratio, rather than the more common vertical format. Microsoft makes a version of its Windows CE operating system specifically for these devices. Known as Handheld PC 2000 (sometimes abbreviated to just H/PC), this comes with pocket versions of popular Microsoft Office applications such as Word and PowerPoint. The major difference from Pocket PC is that the screen resolution is 640x240 pixels—.5 VGA—and that most devices come with a dedicated keyboard. Although not widespread among general consumers, handheld PCs are popular in industrial settings. iTronix makes a rugged, waterproof version for use in harsh environments. Microsoft isn't the only option here; Psion makes a range of consumer devices based on the EPOC operating system. These are mainly popular in the UK and Europe, but they don't seem to have made much impression in the U.S.

Basic PDA Properties

Mobile phones come in a seemingly endless variety of case designs, but their basic underlying characteristics are the same. PDAs, by contrast, come in a wide range of configurations. Models based on Palm OS and Pocket PC have radically different features, but the general package tends to be similar. Because these are meant to be handheld units, most have roughly the same physical dimensions:

- **Connectivity** 9.6 Kbps to 19.2 Kbps CDPD
- **Screen size** 5.7 cm x 5.7 cm (2.25 x 2.25 in.) (Palm); 6 cm x 8 cm (2.25 x 3 in.) (iPAQ)
- **Resolution** 160x160 pixels (Palm) to 240x320 pixels (Pocket PC)
- **Memory** 8MB (Palm) to 32MB (iPAQ)
- **Processing power** 16 MHz (Motorola Dragonball) to 206 MHz (Intel StrongARM)

PDA Connectivity

PDAs have perhaps the widest array of connection options of any mobile wireless device. One very popular wide-area network (WAN) option for any device with a PC card slot is the Sierra Wireless AirCard. As mentioned earlier, several companies also make CDPD modems in the CF format. A pocket PC with an expander slot can take a PCMIA card, making it capable of handling 11 mbps. As a result, Pocket PCs with PCMIA slots can access wireless LANs.

Another option for a mobile connection is to use your mobile phone as an external modem for your PDA. Cables are available to connect several popular mobile phone models to various PDAs. However, this will limit you to the data speed of your mobile phone, typically 9.6 Kbps, and makes your phone unavailable for regular voice calls. Although this is an interim option, or suitable for people who regularly find themselves traveling outside the coverage areas of CDPD, it's likely to become less useful as easier, better integrated wireless solutions become more commonplace.

The first commonly available integrated wireless PDA was the Palm VII, which had a built-in CDPD modem and a flip-up flexible antenna. In the U.S., the Palm VII and VIIx operate over the BellSouth CDPD network, rebranded as Palm.Net, but they are limited to a data speed of about 8 Kbps. The Palm VII is quite a chunky device compared to models like the sleek Palm V. When Palm V users began clamoring for wireless options, a company called OmniSky responded with the Minstrel modem, a thin device that clips onto the back of the Palm. This also uses a CDPD network, although the data speeds are faster than Palm.Net—about 19.2 Kbps. In Europe, Ubinetics markets a similar clip-on for the Palm V that uses GSM, with the maximum data speed limited to about 14.4 Kbps.

Rather than connecting directly to a Palm device, a special proxy server requests content from your site, and then reformats it especially for display on the Palm's limited screen. Pocket PC devices, by contrast, generally make a direct HTTP connection with your Web server. One important point to note is that, regardless of the connection type, the communication with your content Web server is still via conventional HTTP. This is true for Palm Web Clipping, Pocket PC browsers, and WAP phones. There may be intermediate gateway or proxy servers between the device and your server that perform protocol translation, but it is always an HTTP request that is made of your server.

PDA Screen Size

Palm OS–based devices have a screen approximately 5.7 cm x 5.7 cm (2.25 x 2.25), with a resolution of 153 pixels wide by 144 pixels high (the actual screen resolution is 160x160 pixels, but the lower portion is reserved for Palm's handwriting recognition area, and a few pixels at the side are reserved for a vertical scroll bar). Although color models are available, the majority of devices on the market right now are monochrome. Most have a color depth of two bits, meaning they can display only four shades of gray. Although this is a big step up from the tiny WAP phone screen, for a Webmaster designing pages for such a device, this is obviously quite limiting, and you'll need to be creative in how you reformat your pages. Bear in mind also that, prior to Palm OS 4.0, no option for horizontal scrolling was available.

One other device we mentioned earlier, the RIM 957 or Blackberry, also contains a microbrowser. This browser is unique in that it can display both WAP and HTML content. When in HTML mode, it behaves very much like a Palm. In fact, it understands most of the Palm Web Clipping HTML extensions. The screen is also 160x160 pixels, although it can display only black or white. For the most part, you can use exactly the same pages for either Palm or Blackberry devices. The one restriction is that the Blackberry does not use the precompiled graphics capability of Web Clipping. If you're targeting pages for both devices, you'll need to be aware of this.

Some Pocket PC devices, by contrast, have much higher resolution full-color screens. Most can display 240 pixels wide by 320 pixels high (.25 VGA). This is obviously much less limiting for a Webmaster. Pocket PC devices include a version of Internet Explorer 3.2, allowing you to create pages that more closely resemble your standard Web site. In fact, if you take care to allow for the smaller screen and slower connection speeds, you can use the same content for both traditional and wireless users.

PDA screens fall somewhere in between a WAP phone and a full-size laptop. Although WAP browsers are available for Palm and Pocket PC, WAP doesn't take full advantage of the larger screens, easier navigation, and availability of color. Conversely, content formatted for a large screen generally won't look good on a PDA. For instance, left-side navigation bars are a common and intuitive interface on conventional Web sites. A typical navigation bar might be 125 pixels wide, leaving the rest of the screen for content. However, on a Palm, this would leave just 28 pixels for content! Moreover, most navigation bars are constructed with nested tables—something not supported in the version of HTML used for Palm Web Clipping. Although Internet Explorer does a much better job of displaying regular Web sites on Pocket PC, it generally still requires an excessive amount of horizontal and vertical scrolling.

Handheld PCs and devices such as the Psion Revo have a horizontal screen. The Revo and other models have relatively low-resolution monochrome LCD screens, whereas most devices running Microsoft Handheld PC (H/PC) 2000 have full color screens capable of 640 x 240 pixel resolution. A typical H/PC device screen is 16.5 cm (6.5 in.) wide.

PDA Memory

Most Palm OS devices top out at 8MB of memory, and quite a few still get by on just 2MB. Pocket PC devices, by contrast, usually have at least 16MB of RAM. Many have 32MB, and Compaq's new iPAQ 3670 model comes standard with 64MB. Most Pocket PC devices feature an expansion slot that can accommodate extra memory. The two most common expansion options on Pocket PC devices are PCMCIA (the same PC card slot found on all laptops) and CF.

CF memory modules are available in various sizes, from 8MB up to 256MB. IBM even makes its 1GB MicroDrive in CF; rather than the solid-state memory of most CF cards, this is actually a miniscule spinning hard drive. Because CF is a popular storage option for many digital cameras, this makes it easy to move digital images from camera directly to PDA. You can also insert CF cards into laptops, or a PDA with PC card slot, by using a cheap adapter.

Some PDAs accept PCMCIA cards, either directly or via an expansion sleeve. Because this is exactly the same slot found on all laptops, this means that you can use the same cards on both, as long as the manufacturer provides a Pocket PC driver. IBM makes the MicroDrive in this format, which is basically a miniaturized spinning hard disk on a PC card, in capacities from 500MB to over 2GB. If you need to transport large amounts of data on your PDA, and you regularly exchange this data with a laptop, PC card storage is a good and cost-effective option.

Another storage option becoming more popular is the Secure Digital (SD) card. The newer Palm models accept this format, as do several digital cameras and other devices. These are similar to CF, but much smaller—not much bigger than a postage stamp. They come in various denominations, currently available up to 64MB.

PDA Processing Power

Due to the simplicity and efficiency of the Palm OS, these devices are able to perform adequately with relatively slow processors. However, as Palm devices are called on more and more to perform as sophisticated enterprise tools, there's a need to bump up the power. Motorola has announced that they will be doubling the power of the Dragonball chips used in all Palm devices.

Pocket PCs generally come with much more processing power. The OS itself requires more power, but these devices were designed from the outset to perform much more powerful onboard processing tasks. Even the slowest models come with a 66 MHz CPU. The Compaq iPAQ features a powerful Intel StrongARM chip that runs at 206MHz. Table 1.1 lists the processor speed and memory specifications of several popular PDAs.

Table 1.1 CPU Speeds and Memory of Mobile Devices

Maker	Device	Processor	Memory	Speed
Palm	Palm VII	Motorola Dragonball	2MB	16 MHz
Palm	Palm Vx	Motorola Dragonball EZ	8MB	20 MHz
Palm	Palm m505	Motorola Dragonball	8MB	33 MHz
Handspring	Visor Prism/ Platinum	Motorola Dragonball VZ	8MB	33 MHz
Compaq	iPAQ 3650	Intel StrongARM	32MB	206 MHz
Sony	Clié (with Palm OS 4.00)	Motorola Dragonball VZ	8MB	33 MHz
IBM	Workpad	Motorola Dragonball EZ	4MB	16 MHz
Hewlett Packard	Jornada 548	Hitachi SH3	32MB	133 MHz
Symbol	PPT 2800 (Pocket PC)	Intel SA1100	32MB RAM / 32MB ROM	206 MHz
Symbol	SPT 1733 (Palm OS)	Motorola Dragonball	8MB	16 MHz

Laptop Computers

Laptops have been mobile from the beginning, but they have only recently acquired the capability to be wireless. This is a natural fit. Business travelers typically find themselves spending quite a lot of time in places such as airports, on trains, or in hotel rooms lacking a phone jack. The availability of a wireless connection in these areas immediately makes the time spent there more productive. A traveler can download e-mail or the latest Powerpoint presentation before boarding a plane, work on it en route, then upload again upon touching down.

Several manufacturers, such as IBM, HP, and Apple, have begun shipping laptops with built-in wireless LAN (802.11b) cards, with antennas integrated into the casing. These same manufacturers will soon begin offering Bluetooth-equipped laptops. As Wireless LAN and Bluetooth networks become more common in public spaces, it may soon be possible, in airports and large metro areas, to remain constantly connected to the Internet or your corporate systems, wirelessly. Several major airlines have recently announced plans to provide onboard wireless access, so even the time spent in the air may soon be connected.

The recent introduction of more powerful Pocket PC–powered PDAs opens the possibility that many mobile professionals will forego entirely the weight and bulk of laptops in favor of a wireless PDA, perhaps with a lightweight portable keyboard, as their only computer. Another OS option, Handheld PC from Microsoft, is a step up from Pocket PC, and offers functionality closer to a full-blown laptop system, but in a much more portable, instant-on package. With larger screens, keyboards, and more memory and storage, Handheld PCs are beginning to offer a viable alternative to bulky laptops.

Basic Laptop Properties

Laptops are an established product, and everyone from college students to traveling professionals now uses them. Increasingly, both groups are adding wireless communications options. Although larger and heavier than PDAs, they have the advantage of more power, memory, and storage. Because most all laptops feature at least one PCMCIA (PC Card) slot, this has been the natural way to add wireless connectivity.

- **Connectivity** 9.6 Kbps (mobile phone) to 19.2 Kbps (CDPD) or 11 Mbps (wireless LAN)
- **Screen size** Typically a minimum of 25 cm (10 in.) wide
- **Resolution** Minimum 640x480, usually 800x600 pixels and higher
- **Memory** Typically 64MB to 128MB
- **Processing power** Typically 450 MHz to 1 GHz

Laptop Connectivity

Because virtually every laptop comes with at least one PCMCIA slot, this is still the most popular method for adding wireless capability. Although you can purchase a cable to connect your laptop to your mobile phone, just like a PDA, at least in the U.S. this is probably a last resort for those outside the coverage areas of other faster options.

The Sierra Wireless AirCard mentioned earlier gives you a wireless connection rated at 19.2 Kbps in most metropolitan areas. Another advantage of the AirCard, or similar models such as the Novatel Merlin, is that you can eject the card from your laptop and stick it straight into your PDA. As with the PDA, a variety of service plans are available, from flat-rate monthly plans to usage-based plans that bill according to how much data you download. In Europe, travelers can get wireless connections of up to 28.8 Kbps using the Nokia Card Phone or a Ubinetics card on the GSM network in their PC Card slot.

Several manufacturers also make Wireless LAN PC cards, allowing you to access your company or home network wirelessly. As mentioned earlier, Wireless LANs are becoming increasingly common in the places traveling professionals tend to congregate. As these connection points become more widespread, and corporate MIS departments implement Virtual Private Networks (VPNs), it may soon be possible for mobile professionals to access their corporate network, at LAN speeds, from almost anywhere in most major cities.

Although you can detect the type of device accessing your site, in most cases you won't be able to infer the speed of a visitor's connection from this. Just because your visitor is using a 5.0 browser, doesn't mean they're on a high-speed wired connection. You therefore need to make sure that your content is optimized for low-bandwidth connections. To add to the complication, some desktop browsers can also view WML content. The Opera browser can view both HTML and WML, and Klondike is a dedicated WAP browser for desktop PCs.

Laptop Screen Size, Memory, and Processing Power

Most laptops on the market today have a color screen that allows for at the very minimum 800x600 pixel resolution, with most newer models showing 1024x768 pixels. You can generally rely on them having enough memory and processing power to run browser-based applications, so from a Webmaster's perspective, you don't need to do anything special to serve content to laptops, apart from being aware that their connection speed may be low.

Convergent and Future Mobile Wireless Devices

The devices we've mentioned so far are all basic variations on established device families—the mobile phone, the PDA, the laptop. As we've shown, each has inherent limitations for mobile wireless use. Next up the rung are convergent devices that seek to merge aspects of each technology. The first devices to take advantage of GPRS will be hybrid phone/PDA units, similar to those currently available in Europe from Mitsubishi and Sagem, or the Kyocera Smartphone sold in the U.S. Although these hybrids can sometimes compensate for the limitations of single devices, they are still essentially old technology. The Swiss Army Knife approach of trying to force more and more functions into a single device will rapidly run up against basic physical restrictions in devices meant to be small, mobile, and easy to use. Industrial design guru Donald Norman envisions a key-ring-style solution, where you would have a basic communications module, then add other modules as needed for a desired task. Bluetooth and its concept of a personal area network could very well be the enabling technology that makes this a reality. This would also allow the form of devices to more closely match their function. For instance, most people want a mobile phone to be small, light, and simple to use. Bolting on a PDA seriously compromises all of these attributes. But because your phone is basically a communications module, why not have your PDA use it to provide a wireless connection, using Bluetooth to communicate between the two?

As we move into the future, we can expect to see a variety of convergent devices and technologies. At present, a few mobile phone models break with the traditional vertical form factor of the typical mobile phone. Nokia's Communicator looks at first like a rather chunky phone, but the entire front swings open like a clamshell to reveal a larger horizontal screen and a full, although miniaturized, keyboard. This phone's microbrowser can display both WML and HTML content. A newer version, the Communicator 9210, features a color screen and runs the EPOC operating system from Symbian, bringing it closer to a PDA.

Ericsson's R380 also has a larger screen. When closed, only one end is visible, making it look like a traditional phone, but swing the keypad open, and the display switches immediately to a horizontal, touch-sensitive screen with an intuitive graphical user interface operated with a stylus.

The Kyocera Smartphone takes an innovative approach of combining a phone with a Palm device. When closed, the top half of the screen is visible, and looks like a regular—if somewhat large—phone. But swinging the keypad down reveals a full, but slightly reduced, Palm handheld screen.

Another technology that looks set to change the landscape of mobile phones is Microsoft's new Stinger phone. Microsoft is not making the actual hardware itself, leaving that to several major OEMs, but they have designed a totally new operating system for them. Stinger is first and foremost a mobile phone, with every feature of the interface optimized for one-handed use. But it will also allow users to access the Internet, as well as connect directly to their Outlook inboxes. Stinger relies heavily on Microsoft's Mobile Information Server (MIS), a new enterprise server application designed specifically for the mobile Internet. One interesting feature of MIS that attempts to compensate for the slow data speeds of current networks is the ability to have the server selectively remove certain portions of a piece of content, thus cutting down on the amount of data sent. For instance, if you're on a particularly slow connection and in a hurry, you might choose to have it remove all articles and prepositions from your e-mail messages.

It's even possible to remove all white space. While this might initially sound bizarre, condensed e-mails are actually quite readable, at least enough to decide whether you want to download the entire message.

Optional modules are also available for the Handspring Visor PDA that allow you to attach a microphone and speaker to the device, and use it as a phone. As PDAs become more expandable, we can reasonably assume that, provided there is demand for it, more devices will be capable of this cross-functionality.

Occupying a space somewhere between the laptop and a PDA, the Tablet PC is a recent arrival on the scene. Wireless by nature, this is basically a large, touch-sensitive screen, roughly the size of a small laptop screen, and less than an inch thick. Data input is via either handwriting recognition, or an on-screen virtual keyboard. Although this device was initially introduced almost a decade ago, it failed then due to a lack of applications, low power, and user-interface problems. With recent advancements in both processors and memory, and new operating systems optimized for the form factor, the tablet is poised to become popular with traveling professionals.

Although all of these devices are innovative in their own way, they still share one underlying paradigm; communication is through visual display, with feedback and interaction with the user interface via touch screen or keypad. This method of interaction, however, isn't optimal or convenient for people who are actually mobile—in motion—as they use the device. One interface technology that looks set to change the face of mobile computing is voice recognition and synthesis. Voice Markup Language (VoiceML) is an XML-compliant markup language that, used in conjunction with a voice server, can enable people to interact with Web sites through voice alone.

Further down the line, no one can say what new devices may emerge once high-speed wireless networks become ubiquitous, and users come to expect and rely on constant connectivity. In several countries, wireless Internet users already outnumber wired users. As Moore's Law continues to drive the size and cost of computing power down, more and more devices—such as cars, refrigerators, utility meters, and personal music players—will begin to sport wireless Net connections. All of these devices will introduce their own unique capabilities and interface requirements to the wireless Web mix. It's impossible to say where this may lead, but one thing is certain: The job of the wireless Webmaster will continue to be new, exciting, and challenging.

Introducing the Microsoft Mobile Internet Toolkit

One of the challenges mobile application developers face is the different look and feel of applications on different devices. An application that runs perfectly on one device may be totally unusable in another. Not only that, developers often have to learn new languages such as the Wireless Markup Language (WML), Handheld Device Markup Language (HDML) and Compact HTML (cHTML). To cater to the vast array of mobile devices available in the market (PDAs, mobile phones with WAP microbrowsers, and so on), developers can either create multiple sets of documents or use the complex XML Stylesheet Language Transformation (XSLT) to tailor their content for multiple devices. But using XSLT is not a trivial solution, it's a task requiring programmers with good programming foundation.

To solve this customization problem, several solutions were proposed. The most popular one is to write your application in a proprietary language. During runtime, an application server would translate your application into the desired output language. Although such a solution is feasible, developers often have to learn additional languages, thereby increasing the cost and duration of projects.

Together with the introduction of the .NET Framework, Microsoft announced the Microsoft Mobile Internet Toolkit (formerly known as the .NET Mobile Web SDK). With the Microsoft Mobile Internet Toolkit, developers can now leverage on their core strengths in Web development using technologies like ASP.NET and ADO.NET to develop compelling mobile applications. Now, you don't need to customize the output of your application anymore; the .NET mobile framework will take care of it, leaving developers to focus on the business logic of the application. What's more, the toolkit is integrated with Visual Studio.NET, giving you the familiar platform to develop mobile applications.

Overview of the .NET Mobile Architecture

The Mobile Internet Toolkit is built on the Microsoft ASP.NET Web Forms (which we discuss in more detail in Chapter 3). Just as the ASP.NET WebControls provide an improved level of cross browser platform independence in your applications, the Mobile Controls give you an improved level of cross-mobile, devie platform independence. The toolkit includes a set of Mobile Controls that is executed by the Mobile Internet Controls runtime during the execution phase. The key feature of the runtime is its ability to recognize the different types of devices accessing the forms and to generate dynamically the codes that the device can understand.

The toolkit supports such controls as *Calendar*, *Label*, *SelectionList*, and *Textbox*. Because the toolkit is an extension of ASP.NET Web Forms, it supports languages like VB.NET, C#, and JScript.NET. For the rest of this chapter, we give you an insight into the features supported by the Microsoft Internet Toolkit. Chapter 5 covers all these features in greater detail.

Devices Supported by the Microsoft Mobile Internet Toolkit

According to Microsoft, the Microsoft Mobile Internet Toolkit SDK has been tested with the following devices:

- Mitsubishi T250
- Nokia 7110
- Pocket PC with Microsoft Pocket Internet Explorer version 4.5
- Siemens C-35i

The following additional devices and simulators have had limited testing:

- Ericsson R380
- Microsoft Internet Explorer 5.5
- Microsoft Internet Explorer 6.0
- Mitsubishi D502i
- NEC N502i
- Nokia 6210
- Palm VIIx
- Palm V
- Panasonic P502i
- RIM Blackberry 950
- RIM Blackberry 957
- Samsung 850
- Siemens S-35i
- Sprint Touchpoint phone

Developing Mobile Web Forms

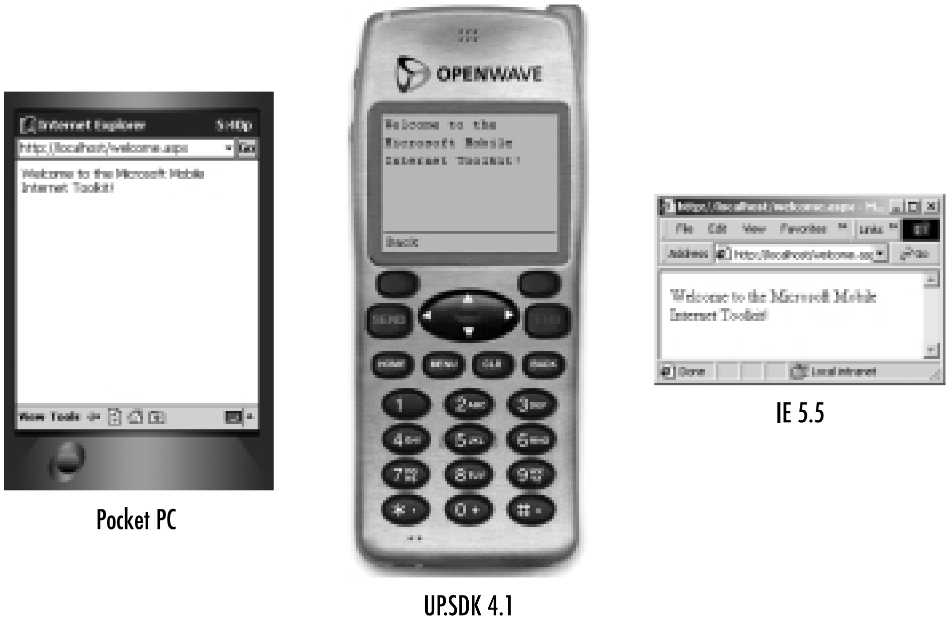
Figure 1.1 gives you a look at a very simple Mobile Web Form and the components it contains.

Figure 1.1 Welcome.aspx

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage"
    Language="VB" %>
<%@ Register TagPrefix="Mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>

<Mobile:Form id="FormOne" runat="server">
   <Mobile:Label runat="server">Welcome to the Microsoft Mobile
       Internet Toolkit!</Mobile:Label>
</Mobile:Form>
```

Figure 1.2 shows the code from Figure 1.1 displayed in different kinds of browsers: Pocket PC, UP.SDK 4.1, and IE 5.5.

Figure 1.2 Viewing the Mobile Web Form on Various Devices



Welcome to the Microsoft Mobile Internet Toolkit!

NOTE

We use a text editor to create the mobile applications in this chapter. Chapter 5 illustrates how you can use Visual Studio.NET to create the same application.

If you have been developing WAP applications using WML and ASP, you would be surprised that the same application can be displayed on all these different devices, with no effort on your side for customization. That's the power of the Microsoft Mobile Internet Toolkit SDK.

Let's now take a closer look at the Welcome.aspx page. The first few lines of a Mobile Web Form contains the standard header directives:

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage"
    Language="VB" %>
<%@ Register TagPrefix="Mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>
```

The *@ Page* directive defines page-specific attributes used by the ASP.NET page parser and compiler. The *Inherits* attribute specifies that the page is inherited from the *System.Web.UI.MobileControls.MobilePage* class, which itself is inherited from the ASP.NET *Page* class. The *Language* attribute specifies the language to be used in the page. This example uses VB.NET. The *@ Register* directive associates prefixes with namespaces and class names. The preceding Mobile Web form used the tag prefix of *Mobile* to associate with the *System.Web.UI.MobileControls* name space. The assembly containing the classes in the namespace you are associating with the given prefix resides is specified in the *assembly* attribute.

Next is the *<Mobile:Form>* element. This element acts as a container to group controls together logically:

```
<Mobile:Form id="FormOne" runat="server">
```

In this case, you have a *<Mobile:Label>* control, which simply provides a label for text to be displayed:

```
   <Mobile:Label runat="server">Welcome to the Microsoft Mobile
       Internet Toolkit!</Mobile:Label>
```

That's all there is to it. As you can see, during runtime when the form is requested, the .NET runtime will automatically detect the type of devices requesting that page and perform a dynamic generation of the target markup languages. In this case, the Pocket PC and IE 5.5 both receive HTML (as shown in Figure 1.3), and the UP.SDK receives WML (as shown in Figure 1.4).

Figure 1.3 HTML Received by Pocket PC and IE 5.5

```
<html><body><form id="FormOne" name="FormOne" method="post"
action="welcome.aspx?__ufps=631315933684236256">
<input type="hidden" name="__EVENTTARGET" value="">
<input type="hidden" name="__EVENTARGUMENT" value="">
<script language=javascript><!—
function __doPostBack(target, argument){
  var theform = document.FormOne
  theform.__EVENTTARGET.value = target
  theform.__EVENTARGUMENT.value = argument
  theform.submit()
}
// —>
</script>
<span>Welcome to the Microsoft Mobile Internet Toolkit!</span><br>
</form></body></html>
```

Figure 1.4 WML Received by the UP.SDK 4.1

```
<?xml version='1.0'?>
<!DOCTYPE wml PUBLIC '-//WAPFORUM//DTD WML 1.1//EN'
'http://www.wapforum.org/DTD/wml_1.1.xml'>
<wml>
   <head>
      <meta http-equiv="Cache-Control" content="max-age=0" />
   </head>
   <card id="FormOne">
      <p>Welcome to the Microsoft Mobile Internet Toolkit!<br/>
      </p>
```

Continued

Figure 1.4 Continued

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

Debugging…

Known Issues with the Pocket PC Emulator

For readers using the Microsoft Embedded Visual Toolkit 3.0, take note that the Pocket PC emulator by default does not support JScript. This feature will pose a problem for mobile applications that make use of JScript for page navigations. To enable JScript support, you should download the JScript.dll component from Microsoft's Web site.

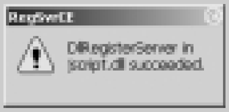
Once the component is downloaded, copy the component into the directory that contains the Embedded Visual Toolkit 3.0 (for example, C:\Windows CE Tools\wce300\MS Pocket PC\emulation\palm300\windows).

After copying the component, open a command window and switch into the directory where the component is copied and execute the following command:

```
C:\Windows CE Tools\wce300\MS Pocket PC\emulation\palm300\
windows>regsvrce jscript.dll
```

If the component is installed correctly, you should see the Pocket PC emulator and the window shown in Figure 1.5.

Figure 1.5 Registering the jscript.dll


RegSvrCE

DllRegisterServer in jscript.dll succeeded.

NOTE

For the remainder of this chapter, we illustrate examples by using the Pocket PC emulator and the UP.SDK 4.1.

Using Multiple Forms in a Single Page

ASP.NET pages can have only a single server-side form; however, you can have multiple mobile forms in a Mobile Web form. In this section, we discuss having multiple forms in a page and how to link them up. Consider the example in Figure 1.6.

Figure 1.6 Multiple_forms.aspx

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage"
    Language="VB" %>
<%@ Register TagPrefix="Mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>

<Mobile:Form id="FormOne" runat="server">
   <Mobile:Label runat="server">This is the first form</Mobile:Label>
   <Mobile:Link runat="server" navigateURL="#FormTwo">Goto Form
       Two</Mobile:Link>
</Mobile:Form>

<Mobile:Form id="FormTwo" runat="server">
   <Mobile:Label runat="server">This is the second form</Mobile:Label>
</Mobile:Form>
```

In Figure 1.6, you have two forms in a page. To link the two forms, use the *<Mobile:Link>* control. The *navigateURL* attribute contains the ID of the form to link to. Note that the ID is preceded by a number character (#).

Linking to Forms on Other Pages

Even though you may have multiple forms on a page, it is common to have forms located in different pages. With Mobile Web Forms, linking to forms on another page is not so straightforward. Consider the two pages in Figures 1.7 and 1.8.

Figure 1.7 Page1.aspx

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage"
    Language="VB" %>
<%@ Register TagPrefix="Mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>
```

Continued

Figure 1.7 Continued

```
<Mobile:Form id="FormOne" runat="server">
   <Mobile:Label runat="server">This is the first form on Page
       1</Mobile:Label>
   <Mobile:Link runat="server" navigateURL="Page2.aspx">Goto Form
       two</Mobile:Link>
</Mobile:Form>
```

Figure 1.8 Page2.aspx

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage"
    Language="VB" %>
<%@ Register TagPrefix="Mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>

<Mobile:Form id="FormTwo" runat="server">
   <Mobile:Label runat="server">This is the second form on Page
       2</Mobile:Label>
</Mobile:Form>

<Mobile:Form id="FormThree" runat="server">
   <Mobile:Label runat="server">This is the third form on Page
       2</Mobile:Label>
</Mobile:Form>
```

The form in the first page (Page1.aspx, Figure 1.8) links to the second page (Page2.aspx, Figure 1.9) by specifying the filename in the *navigateURL* attribute:

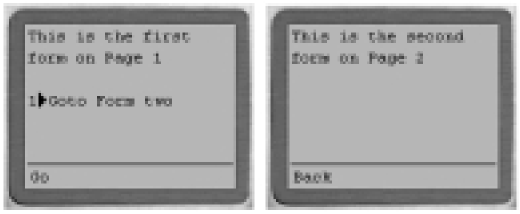
```
   <Mobile:Link runat="server" navigateURL="Page2.aspx">Goto Form
       two</Mobile:Link>
```

This will link to the *FormTwo* on Page2.aspx, as shown in Figure 1.9.

However, if you want to link to *FormThree* in Page2.aspx, you have a problem. Readers who are familiar with WML might recall using the number character (#) to link directly to a card in a deck. So you might have something like the following:

```
<Mobile:Link runat="server" navigateURL="Page2.aspx#FormThree">Goto
    Form two
</Mobile:Link>
```

Figure 1.9 Linking to Forms on Other Pages



This is the first
form on Page 1

1 Goto Form two

Go

This is the second
form on Page 2

Back

However, this is not going to work. It actually turns out to be more involved than just using the #character. Here is the solution:

```
<Mobile:Link runat="server" navigateURL="Page2.aspx?Form=FormThree">
    Goto Form three</Mobile:Link>
```

As shown here, you can add an additional parameter called *Form* and set it to a value of *FormThree*. In Page2.aspx, you can then add this snippet ofVB code:

```
<script runat="server" language="vb">
   Sub Page_Load(sender as Object, e as System.EventArgs)
      Dim formName = Request.QueryString("Form")
      if formName=FormThree.ID then
         ActiveForm = FormThree
      end if
   End Sub
</script>
```

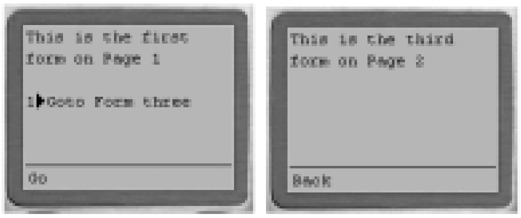
FormThree is now loaded directly (see Figure 1.10).

Dissecting Code

Let's take a more detailed look at the code to see how it works:

```
   Sub Page_Load(sender as Object, e as System.EventArgs)
      Dim formName = Request.QueryString("Form")
      if formName=FormThree.ID then
         ActiveForm = FormThree
      end if
   End Sub
```

Figure 1.10 Jumping Directly to a Form in Another Page



```
This is the first
form on Page 1

1▶Goto Form three

Go
```

```
This is the third
form on Page 2

Back
```

When the page is first loaded, the *Page_Load* event is first triggered. Because the URL contains the parameter *Form*, you can retrieve the value of it using the *Request.QueryString* collection. You can then verify that the value is actually the ID of Form three. If it is, you can then set the current active form to be Form three by using the *ActiveForm* property. The *ActiveForm* property sets and returns the page currently active.

User Inputs

Now that we have looked at linking forms, let's turn our attention to user inputs. The Microsoft Mobile Internet Toolkit supports the following user input controls, with their HTML and WML counterparts shown in Table 1.2:

- *TextBox*
- *Command*
- *List*

Table 1.2 User Input Controls

Input Controls in Mobile Internet Toolkit	Equivalent Tags in HTML	Equivalent Tags in WML
TextBox	<input>	<input>
Command	<input>	<do>
List	<a>	<select><option>

Text and Password Input

To input text into a Mobile Web Form, use the *<Mobile:TextBox>* control provided, as illustrated in Figure 1.11.

Figure 1.11 Password.aspx

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage"
    Language="VB" %>
<%@ Register TagPrefix="Mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>

   <script runat="server" language="vb">
      Sub ComparePassword(sender as Object, e as EventArgs)
         if Password1.Text.Length <8 then
            message.Text = "Password must have at least 8 characters"
            Exit sub
         end if

         if Password1.Text <> Password2.Text then
            message.Text = "Your passwords do not match."
         else
            welcomeMessage.Text += UserName.Text
            ActiveForm = Welcome
         end if
      End sub
   </script>

<Mobile:Form runat="server" id="RegisterForm">
    <Mobile:Label runat="server">Select a user name?</Mobile:Label>
    <Mobile:TextBox runat="server" id="UserName" />
    <Mobile:Label runat="server">Password?</Mobile:Label>
    <Mobile:TextBox password="true" runat="server" id="Password1" />
    <Mobile:Label runat="server">Confirm Password</Mobile:Label>
    <Mobile:TextBox password="true" runat="server" id="Password2" />
    <Mobile:Command runat="server" onClick=
        "ComparePassword">Register</Mobile:Command>
    <Mobile:Label runat="server" id="message"/>
</Mobile:Form>
```

Continued

Figure 1.11 Continued

```
<Mobile:Form runat="server" id="Welcome">
    <Mobile:Label id="welcomeMessage" runat="server">Welcome,
</Mobile:Label>
</Mobile:Form>
```

This page contains two forms, one for allowing the user to input his username and passwords, and one for displaying a message. The output from Figure 1.11 displayed on the Pocket PC and the UP.SDK is shown in Figure 1.12 and Figure 1.13, respectively.

The *<Mobile:TextBox>* control allows text input:

Figure 1.12 User Inputs on Pocket PC

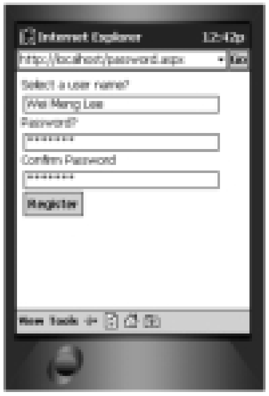
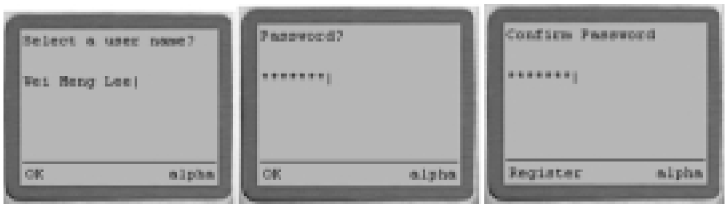

Internet Explorer 12:42p
http://localhost/password.aspx
Select a user name?
Wei Meng Lee
Password?
****
Confirm Password
****
Register

Figure 1.13 User Inputs on UP Emulator


Select a user name?
Wei Meng Lee|
OK alpha

Password?
*******|
OK alpha

Confirm Password
*******|
Register alpha

```
    <Mobile:TextBox runat="server" id="UserName" />
```

To mask the text input (as in the case of entering passwords), specify the *password* attribute as *"true":*

```
    <Mobile:TextBox password="true" runat="server" id="Password1" />
```

You also saw an additional control, *<Mobile:Command>*. The *<Mobile:Command>* control displays a command button so that an action can be performed:

```
    <Mobile:Command runat="server" onClick=
        "ComparePassword">Register</Mobile:Command>
```

The *onClick* attribute indicates the subroutine to call when the user clicks on it. In this case, the subroutine to be invoked is *ComparePassword*.

```
      Sub ComparePassword(sender as Object, e as EventArgs)
         if Password1.Text.Length <8 then
            message.Text = "Password must have at least 8 characters"
            Exit sub
         end if

         if Password1.Text <> Password2.Text then
            message.Text = "Your passwords do not match."
         else
            welcomeMessage.Text += UserName.Text
            ActiveForm = Welcome
         end if
      End sub
```

Within the subroutine, you can simply reference the controls using their IDs. For example, if you want to check for the length of the password that the user has entered, you can simply reference the control using this:

```
Password1.Text.Length
```

If the length of the password is less than eight characters, set the *Text* property of the *Label* control named *message* to contain the following error message:

```
message.Text = "Password must have at least 8 characters"
```

You can also check to see if the two passwords entered are the same. If they are, print a welcome message on the second form:

```
welcomeMessage.Text += UserName.Text
```

The second form is invoked by using the *ActiveForm* property:

```
ActiveForm = Welcome
```

List Selection

Another form of user input is via a selection list. Consider the example in Figure 1.14.

Figure 1.14 Lists.aspx

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage"
    Language="VB" %>
<%@ Register TagPrefix="Mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>

<Mobile:Form runat="server">
   <Mobile:Label runat="server">Membership Types</Mobile:Label>
   <Mobile:List runat="server" id="Membership">
      <Item value="STU" text="Students"/>
      <Item value="PRO" text="Professionals"/>
      <Item value="LIB" text="Libraries"/>
   </Mobile:List>
</Mobile:Form>
```

The *<Mobile:List>* control provides the ability to display lists of items either as a static list or interactive selection. The page in Figure 1.14 causes the screens on the Pocket PC and the UP.SDK (see Figure 1.15 and Figure 1.16, respectively) to be displayed.

Figure 1.15 Viewing the List on the Pocket PC

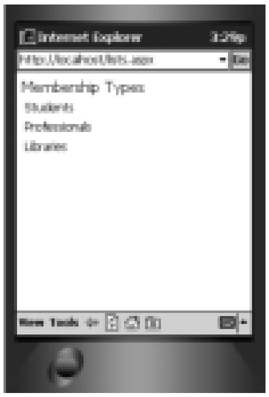
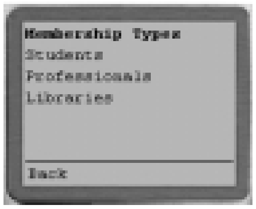

Internet Explorer 3:29p
http://localhost/lists.aspx
Membership Types
Students
Professionals
Libraries

Figure 1.16 Viewing the List on the UP.SDK



Selecting from a List

A static list is not very exciting, not to mention not very useful. A list is useful only if the user can choose from it. The example in Figure 1.17 modifies the previous program to make the list item selectable.

Figure 1.17 Selectlists.aspx

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage"
    Language="VB" %>
<%@ Register TagPrefix="Mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>
<script runat="server" language="vb">
   Sub Select_Item(sender as Object, e as ListCommandEventArgs)
      Dim Fees as integer
      Dim MembershipType as String = e.ListItem.Value
      Select Case MembershipType
         Case "STU"
                    Fees = 38
         Case "PRO"
                    Fees = 95
         Case "LIB"
                    Fees = 1995
      End Select
      FeesPayable.Text = "The fees payable for " & e.ListItem.Text & "
          is $" & Fees
      ActiveForm = FormTwo
   End Sub
```

Continued

Figure 1.17 Continued

```
</script>

<Mobile:Form runat="server" id="FormOne">
   <Mobile:Label runat="server">Membership Types</Mobile:Label>
   <Mobile:List runat="server" id="Membership"
       OnItemCommand="Select_Item">
      <Item value="STU" text="Students"/>
      <Item value="PRO" text="Professionals"/>
      <Item value="LIB" text="Libraries"/>
   </Mobile:List>
</Mobile:Form>

<Mobile:Form runat="server" id="FormTwo">
   <Mobile:Label runat="server" id="FeesPayable" />
</Mobile:Form>
```

Note that this code adds another attribute, *OnItemCommand*, to the *<Mobile:List>* control. This attribute contains the name of the subroutine to be invoked when the list item is selected (see Figure 1.18).

Figure 1.18 List Items Are Selectable

```
   Sub Select_Item(sender as Object, e as ListCommandEventArgs)
      Dim Fees as integer
      Dim MembershipType as String = e.ListItem.Value
      Select Case MembershipType
         Case "STU"
                    Fees = 38
         Case "PRO"
                    Fees = 95
         Case "LIB"
                    Fees = 1995
      End Select
      FeesPayable.Text = "The fees payable for " & e.ListItem.Text & "
          is $" & Fees
```

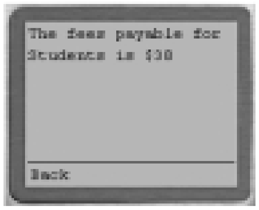
Continued

Figure 1.18 Continued

```
        ActiveForm = FormTwo
    End Sub
```

Within the subroutine, you can use a *Select-Case* statement to find the fees payable; the results are shown in Figure 1.19.

Figure 1.19 Displaying the List Item Selected



```
The fees payable for
Students is $30
```

Data Binding List Items

A list is much more useful if you can dynamically bind it to a list of items. The code in Figure 1.20 illustrates how you can bind a list of items by using the *ArrayList* class in VB.NET.

Figure 1.20 Databind.aspx

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage"
     Language="VB" %>
<%@ Register TagPrefix="Mobile" Namespace="System.Web.UI.MobileControls"
     Assembly="System.Web.Mobile" %>

<script runat="server" language="vb">

    Sub Menu_Item(sender as Object, e as ListCommandEventArgs)
        message.Text = "Fees for " & e.ListItem.Text & " membership is
$" & e.ListItem.Value
        ActiveForm = FormTwo
    End Sub

    Private Class Member
```

Continued

Figure 1.20 Continued

```
     Dim mType as String
     Dim mFees as Single

     Public Sub New(t as String, f as Single)
        mType = t
        mFees = f
     End Sub

     Public Property Type
        Get
           Type = mType
        End Get
        Set
           mType = Value
        end Set
     End Property

     Public Property Fees
        Get
           Fees = mFees
        End Get
        Set
           mFees = Value
        end Set
     End Property
   End Class

   Sub Page_Load (send as Object, e as EventArgs)
      if not (IsPostBack) then
         Dim array as new ArrayList()
         array.Add(new Member("Students", 38))
         array.Add(new Member("Professionals", 95))
         array.Add(new Member("Libraries", 1995))
```

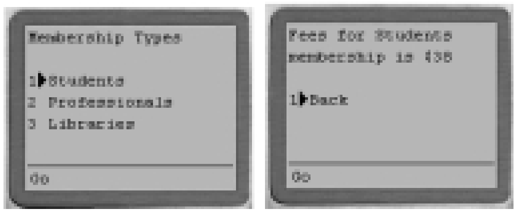
Continued

Figure 1.20 Continued

```
         Menu.DataSource = array
         Menu.DataBind()
      end if
   End Sub

</script>

<Mobile:Form runat="server" id="FormOne">
   <Mobile:Label runat="server" id="test">Membership
       Types</Mobile:Label>
   <Mobile:List runat="server" id="Menu" DataTextField="Type"
       DataValueField=
    "Fees" onItemCommand="Menu_Item"/>
</Mobile:Form>

<Mobile:Form runat="server" id="FormTwo">
   <Mobile:Label runat="server" id="message"/>
   <Mobile:Link runat="server" navigateURL=
      "#FormOne">Back</Mobile:Link>
</Mobile:Form>
```

When the page is loaded, the result is the screen shown in Figure 1.21.

Figure 1.21 Data Binding a List



```
Membership Types

1 Students
2 Professionals
3 Libraries

Go
```

```
Fees for Students
membership is $38

1 Back

Go
```

Dissecting the Code

You first create an array (using the *ArrayList* class) when the page is loaded. An *ArrayList* class is a single dimensional array that can grow dynamically when elements are added to it:

```
Sub Page_Load (send as Object, e as EventArgs)
    if not (IsPostBack) then
        Dim array as new ArrayList()
        array.Add(new Member("Students", 38))
        array.Add(new Member("Professionals", 95))
        array.Add(new Member("Libraries", 1995))
```

In this case, you add three *Member* objects to the array. Once the objects are added to the array, you bind the array to the list:

```
        Menu.DataSource = array
        Menu.DataBind()
```

You may have noticed this line:

```
    if not (IsPostBack) then
```

The *IsPostBack* property contains a Boolean value that indicates whether the page is loaded in response to the client's postback, or if the page is loaded for the first time. The *IsPostBack* property will be true if the user clicks on the *Back* link to return to the main page. You want to make sure that the array is not re-created when the user posts back the page (though it is harmless in this case to re-create the array).

NOTE

The .NET Framework automatically sets the *IsPostBack* property. The programmer does not need to set it.

The *<Mobile:List>* control also contains two additional attributes—*DataTextField* and *DataValueField*:

```
    <Mobile:List runat="server" id="Menu" DataTextField=
        "Type" DataValueField="Fees"
 onItemCommand="Menu_Item"/>
```

The *DataTextField* attribute binds the *Type* property of the *Member* class to the list item's *Text* property. The *DataValueField* attribute binds the *Value* property of the *Member* class to the list item's *Value* property. This is evident from the following line:

```
message.Text = "Fees for " & e.ListItem.Text &
  " membership is $" & e.ListItem.Value
```

Events

Mobile controls (like any other ASP.NET server controls) respond to events. You have seen the various events associated with the controls shown in the earlier examples, for example the following:

```
<Mobile:Command runat="server" onClick=
    "ComparePassword">Register</Mobile:Command>
```

In this example, the *onClick* attribute represents the *onClick* event. The *ComparePassword* subroutine is invoked when the command button is clicked. In this case, the event is related to the control. Page-level events are also available. Look at this next line as an example:

```
Sub Page_Load(sender as Object, e as System.EventArgs)
```

In this case, the event (*Page_Load*) is fired when the page is loaded. Form-level events are also possible using the *OnActivate* attribute of the *<Mobile:Form>* control. To see the sequence in which these two events are fired, consider Figure 1.22.

Figure 1.22 OnActivate.aspx

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage"
    Language="VB" %>
<%@ Register TagPrefix="Mobile" Namespace="System.Web.UI.MobileControls"
     Assembly="System.Web.Mobile" %>


<script runat="server" language="vb">
   Sub Page_Load(sender as Object, e as System.EventArgs)
      message.Text += "Page Loaded. "
   End Sub
```

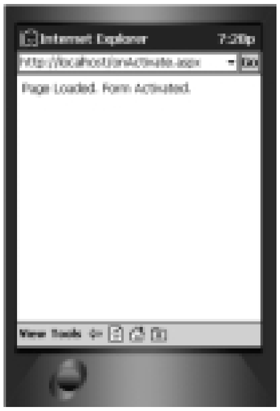
Continued

Figure 1.22 Continued

```
   Sub Form_Activate(sender as Object, e as EventArgs)
      message.Text += "Form Activated. "
   End Sub
</script>

<Mobile:Form id="FormOne" runat="server" onActivate="Form_Activate">
   <Mobile:Label runat="server" id="message"/>
</Mobile:Form>
```

When the page in Figure 1.22 is loaded, the screen shown in Figure 1.23 is displayed.

Figure 1.23 Demonstrating the Sequence of Events



Page Loaded. Form Activated.

Thus, you can see that the *Page_Load* event is fired first, followed by the *OnActivate* event of the *<Mobile:Form>* control.

Displaying Images

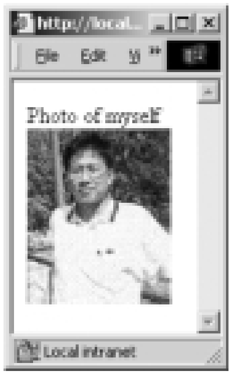
To display images, you can use the *<Mobile:Image>* control. Because various mobile devices display images of differing format, you need to send the correct image type to the right device. To solve this problem, you can use the *<DeviceSpecific>* control, as shown in Figure 1.24.

Figure 1.24 Image.aspx

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage"
    Language="VB" %>
<%@ Register TagPrefix="Mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>

<Mobile:Form runat="server">
   <Mobile:Label>Photo of myself</Mobile:Label>
   <Mobile:Image runat=server alternateText="[My Photo here]">
    <DeviceSpecific>
      <Choice Filter="isHTML32" ImageURL="myself.bmp" />
      <Choice Filter="isWML11"  ImageURL="myself.wbmp" />
    </DeviceSpecific>
   </Mobile:Image>
</Mobile:Form>
```

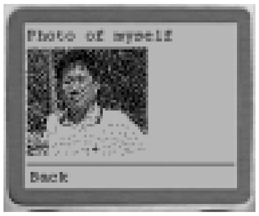
Within the *<DeviceSpecific>* control, you have the *<Choice>* elements. In the preceding program, each choice element contains two attributes: *Filter* and *ImageURL*. So in this case, if the user were using a Web browser, the BMP file would be displayed, as shown in Figure 1.25.

Figure 1.25 Displaying the BMP File in a Web Browser



On the UP.SDK, the WBMP file would be selected, as shown in Figure 1.26.

Figure 1.26 Displaying the WBMP File in a WAP Browser



The *<Choice>* elements are evaluated according to the order in which they appear in the *<DeviceSpecific>* control. If none of the *<Choice>* elements evaluates true, the string "[My Photo here]" would be displayed. The *Filter* attribute contains values that are matched from the *<deviceFilters>* element in the web.config configuration file (see Figure 1.27):

```
<Choice Filter="isHTML32" ImageURL="myself.bmp" />
<Choice Filter="isWML11"  ImageURL="myself.wbmp" />
```

Figure 1.27 Web.config

```
<deviceFilters>
   <!— Markup Languages —>
   <filter name="isHTML32" compare="preferredRenderingType"
                              argument="html32" />
   <filter name="isWML11" compare="preferredRenderingType"
                              argument="wml11" />
   <filter name="isCHTML10" compare="preferredRenderingType"
                              argument="chtml10" />
   <!— Device Browsers —>
   <filter name="isGoAmerica" compare="browser"
                              argument="Go.Web" />
   <filter name="isMME" compare="browser"
                              argument="Microsoft Mobile Explorer" />
   <filter name="isMyPalm" compare="browser"
                              argument="MyPalm" />
   <filter name="isPocketIE" compare="browser"
                              argument="Pocket IE" />
```

Continued

Figure 1.27 Continued

```
    <filter name="isUP3x" compare="type"
                              argument="Phone.com 3.x Browser" />
    <filter name="isUP4x" compare="type"
                              argument="Phone.com 4.x Browser" />
    <!— Specific Devices —>
    <filter name="isEricssonR380" compare="type"
                              argument="Ericsson R380" />
    <filter name="isNokia7110" compare="type"
                              argument="Nokia 7110" />
    <!— Device Capabilities —>
    <filter name="prefersGIF" compare="preferredImageMIME"
                              argument="image/gif" />
    <filter name="prefersWBMP" compare="preferredImageMIME"
                              argument="image/vnd.wap.wbmp" />
    <filter name="supportsColor" compare="isColor"
                              argument="true" />
    <filter name="supportsCookies" compare="cookies"
                              argument="true" />
    <filter name="supportsJavaScript" compare="javascript"
                              argument="true" />
    <filter name="supportsVoiceCalls" compare="canInitiateVoiceCall"
                              argument="true" />
</deviceFilters>
```

The web.config file contains the various device filters. Figure 1.27 shows a portion of the web.config file. A device may match several *<filter>*s. For example, a Web browser satisfies the *isHTML32* and the *prefersGif* filter. The *compare* attribute of the *<filter>* element specifies the capability evaluated by the comparison evaluator and the *argument* attribute specifies the argument against which the capability should be compared. To illustrate using the preceding example, a WAP device will match the *isWML11* filter, which will in turn match the second *<Choice>* element.

Validation Controls

Quite a few validation controls are available in the Microsoft Mobile Internet Toolkit SDK:

- **CompareValidator** Compares two controls using a specified operator.
- **CustomValidator** Allows customized validation of controls.
- **RangeValidator** Validates the value of a control to ensure that it falls within a specified range.
- **RegularExpressionValidator** Validates the value of a control by specifying a regular expression.
- **RequiredFieldValidator** Ensures that a field is supplied a value.
- **ValidationSummary** Displays the summary of all errors that occurred during the rendering of a form.

To see how they work, consider the example shown in Figure 1.28.

Figure 1.28 Validation.aspx

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage"
    Language="VB" %>
<%@ Register TagPrefix="Mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile"
%>

<script language="vb" runat=server>
   Sub Submit_OnClick(sender as Object, e as EventArgs)
      if (Page.IsValid) then
         ActiveForm = Form2
         Result.Text = "The month you have entered was " & month.Text
      end if
   End sub
</script>

<Mobile:Form id="Form1" runat=server>
    <Mobile:RangeValidator runat=server
       ControlToValidate="month"
```

Continued

Figure 1.28 Continued

```
       Type="Integer"
       MaximumValue="12"
       MinimumVaLue="1">
       The month is not correct. Please try again.
    </Mobile:RangeValidator>
    <Mobile:Label runat=server>Please enter your birth
        month</Mobile:Label>
    <Mobile:TextBox id="month" Numeric="true" runat=server/>
    <Mobile:Command OnClick="Submit_OnClick"
        runat=server>Submit</Mobile:Command>
</Mobile:Form>

<Mobile:Form id="Form2" runat=server>
    <Mobile:Label id="Result" runat=server/>
    <Mobile:Link Text="Back" navigateURL="#Form1" runat=server/>
</Mobile:Form>
```

The preceding example uses the *<Mobile:RangeValidator>* control to validate the range of a number:

```
    <Mobile:RangeValidator runat=server
       ControlToValidate="month"
       Type="Integer"
       MaximumValue="12"
       MinimumVaLue="1">
       The month is not correct. Please try again.
    </Mobile:RangeValidator>
```

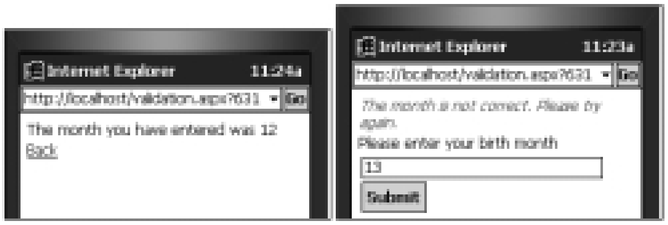
Once the number is entered and the button is clicked, the *Submit_OnClick()* subroutine is invoked. The *IsValid* property will validate the range of the number entered. If the validation fails, the message "The month is not correct. Please try again" is displayed; otherwise Form2 will be loaded:

```
   Sub Submit_OnClick(sender as Object, e as EventArgs)
      if (Page.IsValid) then
         ActiveForm = Form2
```

```
        Result.Text = "The month you have entered was " & month.Text
    end if
End sub
```

Figures 1.29 and 1.30 show the output as displayed by the Pocket PC emulator and the UP.SDK, respectively.

Figure 1.29 Using the Validator Controls on the Pocket PC

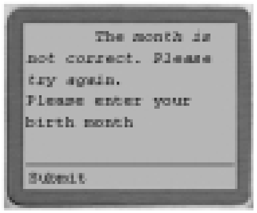

Internet Explorer 11:24a
http://localhost/validation.aspx?631
Go
Back
Internet Explorer 11:23a
http://localhost/validation.aspx?631
Go
13
Submit

The month you have entered was 12

The month is not correct. Please try again.

Please enter your birth month

Figure 1.30 Using the Validator Controls on the UP.SDK



The month is not correct. Please try again.

Please enter your birth month

Paginations

In an earlier section, you saw the use of the *<Mobile:List>* control. Sometimes, the list of items within the control might be long. Anyone who has written a WAP application can attest to the importance of keeping the list short, at least per screen. A common technique is to display the list in multiple pages, and as such this technique is commonly known as *records paging*. One of the great features of the Mobile API is its auto-paging capability. Consider the example shown in Figure 1.31.

Figure 1.31 Paginate.aspx

```
<%@ Page Inherits="System.Web.UI.MobileControls.MobilePage"
    Language="VB" %>
<%@ Register TagPrefix="Mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>
```

Continued

Figure 1.31 Continued

```
<script language="vb" runat=server>
    Sub Select_Item (sender as Object, e as ListCommandEventArgs)


    End Sub
</script>

<Mobile:Form runat="server" id="form1"
                           paginate="true"
                           PagerStyle-NextPageText="Go to Page {0}"
                           PagerStyle-PreviousPageText="Back to Page
{0}">
    <Mobile:Label runat="server" StyleReference="title" Text="Books in
        the .net Developer Series" />
    <Mobile:Label runat="server" id="PageNo"/>
    <Mobile:List runat="server" id="titles"
        OnItemCommand="Select_Item">
       <Item value="1" text="ADO .net Developer's Guide"/>
       <Item value="2" text="Web Services Developer's Guide"/>
       <Item value="3" text="C#.net Developer's Guide"/>
       <Item value="4" text="ASP.net Developer's Guide"/>
       <Item value="5" text=".net Mobile Web Developer's Guide"/>
       <Item value="6" text="VB.net Developer's Guide"/>
       <Item value="7" text="XML Developer's Guide to Web-based EDI"/>
    </Mobile:List>
</Mobile:Form>
```

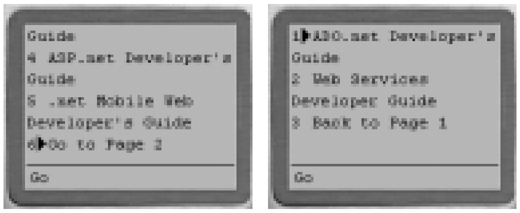
The list contains seven items. When loaded using the UP.SDK, you can see that the list is displayed in multiple cards (see Figure 1.32).

To allow for paging, simply insert the *Paginate* attribute into the *<Mobile:Form>* control and set it to "true". Additionally, the *PagerStyle-NextPageText* and the *PagerStyle-PreviousPageText* attributes allow you to set the message for displaying the next and previous page, respectively:

```
<Mobile:Form runat="server" id="form1"
```

```
                           paginate="true"
                           PagerStyle-NextPageText="Go to Page {0}"
                           PagerStyle-PreviousPageText="Back to Page
{0}">
```

Figure 1.32 Paginating a Form



```
Guide
4 ASP.net Developer's
Guide
5 .net Mobile Web
Developer's Guide
6 Go to Page 2
Go
```

```
1 ADO.net Developer's
Guide
2 Web Services
Developer Guide
3 Back to Page 1
Go
```

Calendar Control

Apart from those regular controls like *Label* and *Textbox*, the Microsoft Mobile Internet Toolkit also includes some interesting controls like the *Calendar* and *AdRotator* controls. We illustrate the use of the Calendar control in this section.

Date selection is a commonly used feature of mobile applications, and in the past, great efforts have gone into making date selection as easy and error-proof as possible. Instead of spending time in building the date selection module, the Mobile API has included the *Calendar* control. Consider the example shown in Figure 1.33.

Figure 1.33 Birthdate.aspx

```
<%@ Page Inherits="System.Mobile.UI.MobilePage" Language="VB" %>
<%@ Register TagPrefix="Mobile" Namespace="System.Mobile.UI" %>

<script language="VB" runat="server">
   Sub date_Changed(sender as Object, e as EventArgs)
      message.Text = "So your birthdate is " & birthdate.SelectedDate
   End Sub
</script>

<Mobile:Form id="Form1" runat="server">
```

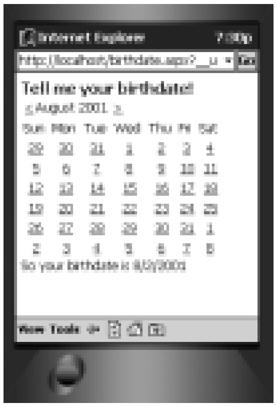
Continued

Figure 1.33 Continued

```
    <Mobile:Label runat="server" styleReference="Title" Text="Tell me
        your birthdate!"/>
    <Mobile:Calendar id="birthdate" OnSelectionChanged="date_Changed"
        runat="server"/>
    <Mobile:Label runat="server" id="message"/>
</Mobile:Form>
```

Figures 1.34 and 1.35 show how the code appears in the various emulators.

Figure 1.34 Using the Calendar Control on the Pocket PC



Internet Explorer 7:30p

http://localhost/birthdate.aspx?__u Go

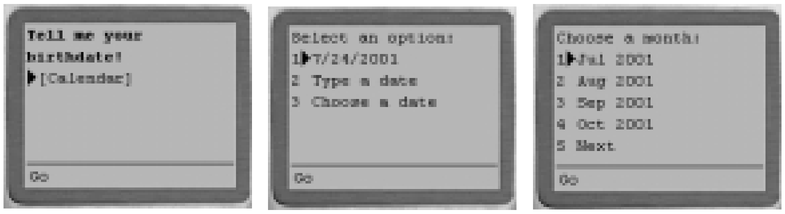
Tell me your birthdate!

< August 2001 >

Sun	Mon	Tue	Wed	Thu	Fri	Sat
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

So your birthdate is 8/3/2001

Figure 1.35 Using the Calendar Control on the UP.SDK



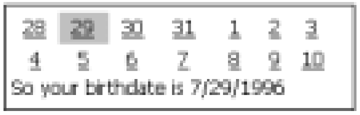
```
Tell me your
birthdate!
▶[Calendar]

Go
```

```
Select an option:
1▶7/24/2001
2 Type a date
3 Choose a date

Go
```

```
Choose a month:
1▶Jul 2001
2 Aug 2001
3 Sep 2001
4 Oct 2001
5 Next

Go
```

When the user selects a date, the message in Figure 1.36 displays.

Figure 1.36 Printing the Birth Date



28	29	30	31	1	2	3
4	5	6	7	8	9	10

So your birthdate is 7/29/1996

If you want the individual day, month, and year printed instead (it is restricted to mm/dd/yyyy format), you can use the following properties:

```
message.Text = "So your birthdate is " & birthdate.SelectedDate.day
```

or

```
message.Text = "So your birthdate is " & birthdate.SelectedDate.month
```

or

```
message.Text = "So your birthdate is " & birthdate.SelectedDate.year
```

Summary

The wireless Web represents a revolution in access to online information that will have profound impact on our society—greater even than the Internet revolution of the past decade. The sheer variety of different mobile devices is the first big change a Webmaster will have to deal with, making the "browser wars" of the last few years seem like child's play. However, you'll find that devices fall into a few general families with similar characteristics, simplifying the presentation task somewhat. Device detection, coupled with on-the-fly, server-side generation of markup is one way to solve this problem.

How users connect to your site can be as important as which device they use. The majority of current mobile devices are limited to wireless speeds of between 9.6 Kbps to 19.2 Kbps, with few exceptions. These connections are also inherently unreliable, with long call setup times and latency. You'll need to modify your content and navigation to compensate for these limitations. The *walled garden* approach of carriers, coupled with the inevitable backlash against WAP when users discovered it didn't live up to the early hype, has left a lot of users disillusioned with WAP, but newly available wireless options for PDAs and laptops have revived interest in mobile computing. With the huge surge in interest in wireless data services—particularly from corporate users—carriers continue to offer newer and faster connectivity options. 2.5G services such as GPRS are already being rolled out worldwide, and we can look forward to the much higher data speeds of 3G introducing a world of broadband wireless within the next two to four years.

The mobile wireless landscape is in a state of rapid flux, with new devices announced almost monthly. Hybrid devices attempt to combine the mobility and voice capabilities of mobile phones with the organizational capabilities of PDAs. Unfortunately, these tend to compromise the capabilities of each. Microsoft Stinger is one device to watch, although its impact is still uncertain. Tablet PC holds some promise as a next-generation mobile device, but is not yet available to end-users. Voice interaction is another emergent technology that could radically affect the job of the wireless Webmaster.

This chapter has covered quite a lot of ground for developers wishing to utilize the Microsoft .NET Framework for mobile application development. In particular, we have introduced you to how ASP.NET works and how you can use the Microsoft Mobile Internet Toolkit and ADO.NET to develop compelling mobile applications.

Because the .NET Mobile Architecture is an extension of the ASP.NET Web Forms, developing mobile applications is very similar to developing Web applications. In the early part of this chapter, we have seen how ASP.NET addresses the problem of HTTP statelessness using HTML and Web Server Controls. Using the architecture in ASP.NET, the Mobile Internet Toolkit allows developers to write mobile applications using the mobile controls, and during runtime it automatically will generate the appropriate codes for the mobile devices.

The set of mobile controls the Mobile Internet Toolkit provides offers rich functionality for mobile developers. The nicest feature is that the runtime frees the developer from the arduous tasks of customizing the application for the myriad devices available in the market.

Solutions Fast Track

Evolving Mobile Devices

☑ The three main categories of mobile devices—mobile phones, PDAs, and laptop computers—are differentiated by connectivity, screen size, memory, and processing power.

☑ Data-capable phones use the WAP protocol, and content needs to be coded in Wireless Markup Language (WML). They have minimal requirements for memory and processing power. A mobile phone never communicates directly with your Web server; a WAP gateway always acts on its behalf (the gateway may alter the content somewhat on its way through).

☑ The market for Personal Digital Assistants (PDAs) is split mainly between those running the Palm operating system from both Palm, Inc. and its licensees, and devices based on Microsoft's Pocket PC or Windows CE. One feature of Pocket PCs that's especially relevant to wireless is that most come with an industry-standard expansion slot, either CF or PCMCIA Type II.

☑ PDAs come in a wide range of configurations of connectivity, screen size, memory, and processing power.

☑ Several manufacturers have begun shipping laptops with built-in wireless LAN (802.11b) cards, with antennas integrated into the casing. These same manufacturers will soon begin offering Bluetooth-equipped laptops. However, with larger screens, keyboards, and more memory and storage, Handheld PCs are beginning to offer a viable alternative to bulky laptops.

☑ Also, several devices are available that seek to combine aspects of each category—a mobile phone with an integrated Palm screen, PDAs that can be used as phones, and laptop-size devices without keyboards that you use by writing directly on the screen.

Introducing the Microsoft Internet Toolkit

☑ The Mobile Internet Toolkit is built on the Microsoft ASP.NET Web Forms and supports languages such as VB .NET, C#, and JScript.NET. It is an extension to the ASP.NET model.

☑ The toolkit includes a set of Mobile Controls that is executed by the Mobile Internet Controls runtime during the execution phase.

☑ The key feature of the runtime is its ability to recognize the different types of devices accessing the forms and to generate dynamically the codes that the device can understand.

Developing Mobile Web Forms

☑ During runtime when the form is requested, the .NET runtime automatically will detect the type of devices (our examples use Pocket PC, IE 5.5 and UP.SDK) requesting that page, and will perform a dynamic generation of the target markup languages. Unlike WAP applications developed using WML and ASP, the same ASP.NET application can be displayed on different devices, with no effort on your part for customization.

☑ ASP.NET pages can have only a single form; however, you can have multiple mobile forms in a Mobile Web form. To link the two forms, you use the *<Mobile:Link>* control. The *navigateURL* attribute contains the ID of the form to link to.

☑ Linking to forms on another page is not so straightforward. The form in the first page links to the second page by specifying the filename in the *navigateURL* attribute. Subsequent steps involve adding another parameter called *Form*, retrieving its value using the *Request.QueryString* collection, verifying the form ID in that value, and using the *ActiveForm* property to set and return the active page.

☑ The Microsoft Mobile Internet Toolkit supports user input controls *TextBox*, *Command*, and *List*.

☑ To input text into a Mobile Web Form, use the *<Mobile:TextBox>* control. To display a command button so that an action can be performed, use the *<Mobile:Command>* control. To display lists of items either as a static list or interactive selection, use the *<Mobile:List>* control. You can also dynamically bind a list of items using the *ArrayList* class.

☑ To display images, you can use the *<Mobile:Image>* control. Because various mobile devices display images of differing format, use the *<DeviceSpecific>* control (within which are the *<Choice>* elements) to send the correct image type to the right device.

☑ Validation controls available in the Microsoft Mobile Internet Toolkit SDK include *CompareValidator*, *CustomValidator*, *RangeValidator*, *RegularExpressionValidator*, *RequiredFieldValidator*, and *ValidationSummary*.

☑ Other features of the Mobile API are its records paging capability, using the *Paginate* attribute, and also its *Calendar* control for date selection.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: What are the main differences between Beta 1 and Beta 2 of the Microsoft Mobile Internet Toolkit?

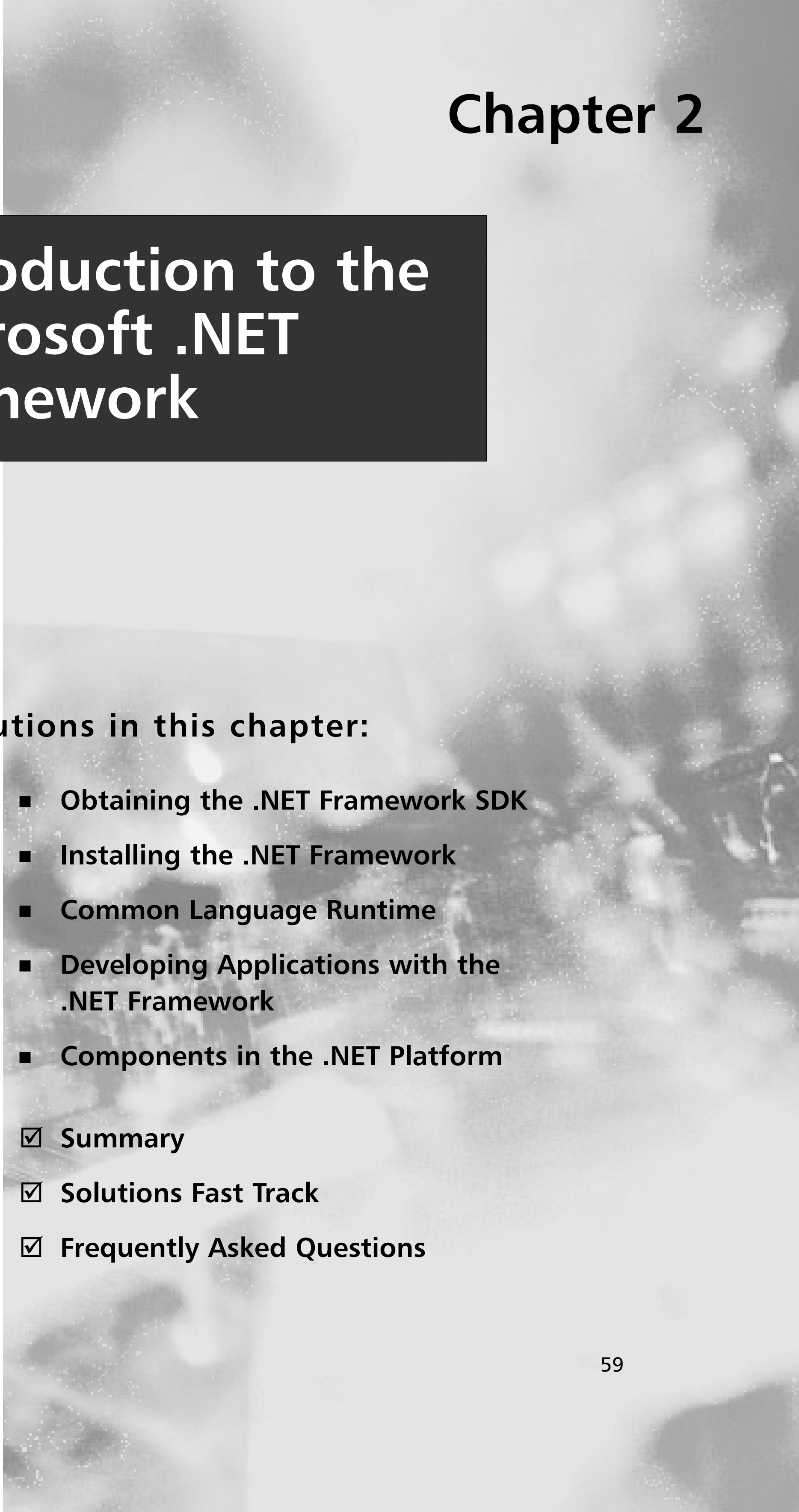
A: Besides the name change, you can now develop mobile application using Visual Studio.NET. Also, more devices are supported in Beta 2. For more information about the changes from Beta 1 to Beta 2, refer to the documentation that comes with the Toolkit.

Q: I am developing WAP applications using the Mobile Internet Toolkit. Do my users still need a WAP gateway?

A: Yes, the Mobile Internet Toolkit simply provides an easier way to develop mobile applications that runs on multiple devices. Your user still needs to use a WAP gateway to access your application.

Q: Do I need to know VB to build for mobile .NET?

A: You can use any language that is compatible with the .NET Common Language Runtime, which includes Microsoft Visual Basic, Microsoft Visual C#, and Microsoft JScript .NET.



Chapter 2

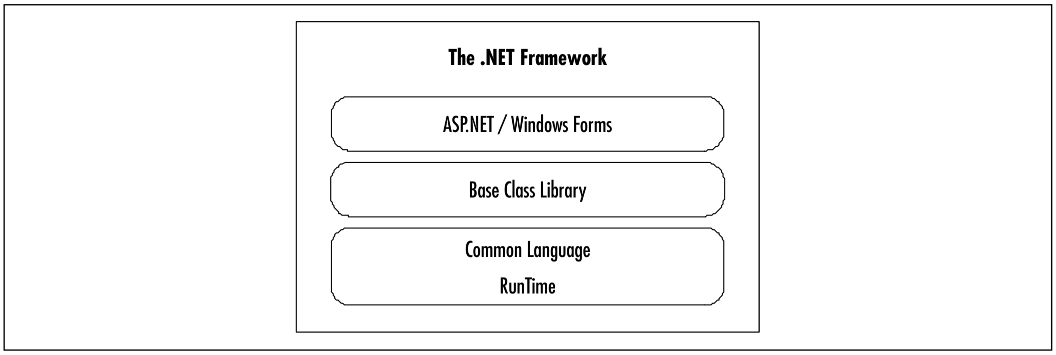
Introduction to the Microsoft .NET Framework

Solutions in this chapter:

Introduction

The .NET Framework is just part of Microsoft's overall .NET platform strategy. The Framework is made up of the Common Language Runtime environment, Base Class Library, and higher-level frameworks such as ASP.NET and Windows Forms, as shown in Figure 2.1.

Figure 2.1 The .NET Framework Architecture



The complete documentation on the .NET Framework fills entire books. What we do here is cover the basics so that you have a firm enough understanding of the .NET Framework to enable you to get started coding using the Mobile Internet Toolkit as soon as possible.

We start with how to obtain the .NET Framework SDK. The minimum system requirements can be confusing, so we cover those in some detail. The Common Language Runtime is the foundation that sits on top of the Windows operating system. Because this is the most important part of the .NET Framework, we spend most of our time going through the Common Language Runtime. The Base Class Library is a set of hundreds of classes that are provided as part of the Framework to help you build applications that will execute in the Common Language Runtime. We give an overview of these so that you can get an idea of the breadth of support provided by the SDK "out of the box." To make building applications even easier, Microsoft provides some higher-level frameworks, such as ASP.NET and Windows Forms, that utilize and extend the functionality provided by the Base Class Library. ASP.NET greatly simplifies the building of Internet applications by using Web Forms and Web services. Windows Forms provides the ability to develop for the rich environment that the Windows platform provides. We take a look at these frameworks to give you an overview of the purpose of each.

Obtaining the .NET Framework SDK

The .NET Framework SDK contains a Framework Runtime for redistribution, compilers, tools, documentation, and samples. This is everything you need to develop, test, and deploy applications that target the .NET Framework. You can obtain the .NET Framework from multiple places. Before detailing those locations, we go over the system requirements for installing the .NET Framework. Please review them carefully, and you will save yourself some major headaches.

We recommend that you install the .NET Framework on a machine or partition that has a new operating system installation and is totally separate from your important applications and data. If you encounter any unforeseen problems, just reformatting and starting over is much easier. This is a good practice with any beta software. At a minimum, you should uninstall any previous versions of the .NET Framework SDK prior to installing a newer version, and don't install on a machine that has production data on it.

NOTE

The .NET Framework SDK and Visual Studio .NET are not the same thing. Visual Studio .NET is an application development tool that enables you to create applications targeted for the .NET Framework Common Language Runtime. You can create .NET applications with any text editor and compile with the SDK provided compilers. Visual Studio .NET just makes it easier.

System Requirements

Prior to installing the .NET Framework SDK, you must make sure that your system meets or exceeds the minimum system requirements. Remember that they are the "minimum" requirements. As always, before you install any subsequent versions, be sure to note the requirements in the documentation provided with the install. Requirements can change from version to version as enhancements are made to the SDK. The following sections outline the .NET Framework SDK minimum system requirements.

Hardware

The hardware requirements are quite reasonable by today's standards for desktops and servers. You can see by the minimum and recommended notes that the more RAM you have, the better.

Desktop .NET Framework SDK Install

The requirements for developing desktop applications using the .NET Framework SDK are:

- **Processor** Intel Pentium class 90 MHz or higher
- **RAM** 32MB (96MB or higher recommended)
- **Video** 800×600, 256 colors

Server .NET Framework SDK Install

The following requirements are to install and develop with the .NET Framework SDK, not just the runtime. We'll cover the requirements for the runtime in the next section.

- **Processor**: Intel Pentium class 133 MHz or higher
- **RAM**: 128 MB (256 MB or higher recommended)
- **Hard disk space required to install**: 360 MB
- **Hard disk space required**: 210 MB
- **Additional space required to install and compile all samples**: 300 MB
- **Video**: 800×600, 256 colors

NOTE

ASP.NET is not supported on Windows 95/98 operating systems such as Windows Me, Windows 98 Second Edition, Windows 98, and Windows 95.

Operating System

Currently, the only operating systems that you use with the .NET Framework SDK are Windows operating systems. Because the Common Language Runtime is so much like the Java virtual machine, don't be surprised if some enterprising third party creates a common language runtime for another operating system. For now, here are the operating system requirements for the .NET Framework development and runtime.

.NET Framework SDK

The following are the operating system requirements for developing with the .NET Framework SDK:

- Windows XP, Windows 2000, Windows NT 4.0
- Internet Explorer 5.01 or later

.NET Framework Redistributable Package (Runtime)

Please note that the following requirements are for the runtime only. They do not include development. You would install the runtime on a machine that contains and runs deployed components. Just as you would expect, the requirements for only the runtime are less restrictive that those for development with the SDK. The operating system requirements for the runtime are as follows:

- Windows XP, Windows 2000, Windows NT 4.0
- Windows 98, Windows Me
- Internet Explorer 5.01 or later

Additional Installation Information

Frequently included with the install is documentation listing any late-breaking issues that have come up since the version was first issued. This is information in addition to the normal readme.htm file. The following list includes a few items that are in the .NET Framework SDK install documentation and late-breaking issues that you may overlook if you're not careful:

- Internet Explorer 5.5 is distributed along with the .NET Framework installation.
- For Server setups, Microsoft Data Access Components (MDAC) 2.6 is required, MDAC 2.7 is recommended. You can download this from www.microsoft.com/downloads.

- For Windows 2000 installs: Internet Information Services (IIS) 5.0 must be installed prior to installing the .NET Framework SDK Beta 2. If IIS is not present on the computer when the SDK installation is performed, ASP.NET will fail to register correctly and ASP.NET applications will not run. If IIS is installed after the SDK, you must manually register the Aspnet_isapi.dll file using Regsvr32.exe.
- For Windows NT 4.0 setups, Service Pack 6a must be installed.

NOTE

Visual Studio .NET includes the .NET Framework SDK.

Locations for Downloading

You can choose among multiple locations for downloading the .NET Framework SDK. Some of these include:

- www.microsoft.com/net
- www.gotdotnet.com
- msdn.microsoft.com/net/

You can also obtain the SDK on a CD by going to the appropriate Microsoft Developer Store, the locations of which are listed on the Web sites noted in the preceding bullet. If you are an MSDN (Microsoft Developer Network) subscriber you can download SDK from the MSDN subscriber download site (msdn.microsoft.com).

Installing the .NET Framework

Because the download file is over 120MB, you have a choice of downloading one large file or many smaller ones. If you decide to go the multifile download route, you need to take the following steps to prepare the install:

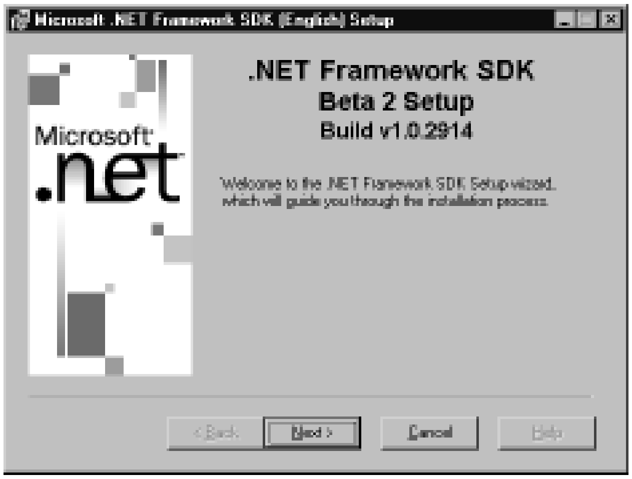
1. Place each downloaded file into the same directory.
2. Run the **setup.bat** file. This will create a master setup.exe file. To find out if your COPY command supports the /Y switch, go to a DOS

prompt and type in **COPY /?**. If the /Y switch is listed, your system supports it. If you don't, you need to edit the setup.bat file to remove the /Y switch from the COPY command.

3. Run the **setup.exe** to install the .NET Framework SDK.

The installation program provides a wizard that takes you through the process, as shown in Figure 2.2.

Figure 2.2 Initial Setup Screen



Microsoft .NET Framework SDK (English) Setup

.NET Framework SDK
Beta 2 Setup
Build v1.0.2914

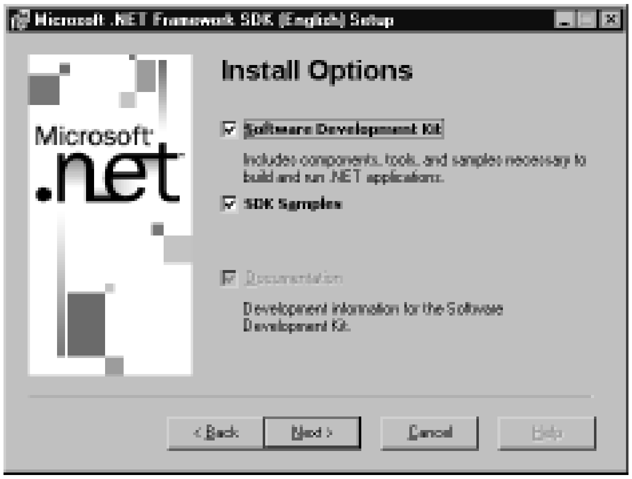
Welcome to the .NET Framework SDK Setup wizard, which will guide you through the installation process.

The installation program searches your machine to see if you have any components already installed and will direct your options accordingly. If you don't have the updated Windows Installer components, you will be asked whether or not you want to update them. Because they are required for the install, you must answer **Yes**.

Figure 2.3 shows an example of an installation that already has the documentation installed. So if this is the first .NET Framework SDK installation on your target machine, you have the option of installing the SDK, the SDK samples, and/or the documentation.

The SDK install includes the Framework as well as ASP.NET. Once you have the install completed, you can verify proper operation by going to the …\Microsoft.NET\FrameworkSDK\Samples\StartSamples.htm page and running some samples. This page has tutorials, technology samples, and a few full applications that will allow you to make sure that you are set up and ready to code. You can get a jump start on your learning as well.

Figure 2.3 .NET Framework SDK Installation Options



☑ Software Development Kit

Includes components, tools, and samples necessary to build and run .NET applications.

☑ SDK Samples

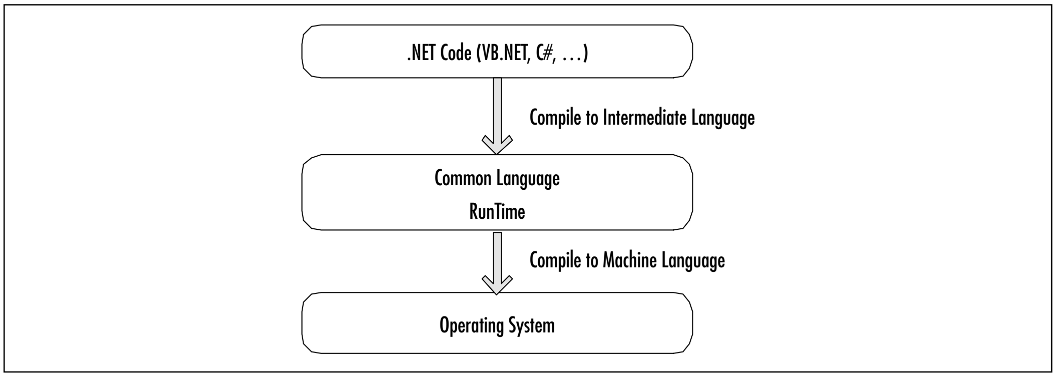
☑ Documentation

Development information for the Software Development Kit.

Common Language Runtime

The Common Language Runtime (CLR) is arguably the most important part of the .NET Framework. It is responsible for executing and managing the compiled output of code written in the .NET-compatible languages. The CLR is the engine at the core of managed code execution. Architecturally, it sits above the operating system and below the .NET language code, as shown in Figure 2.4.

Figure 2.4 CLR Placement in .NET Architecture


.NET Code (VB.NET, C#, …)
Compile to Intermediate Language
Common Language RunTime
Compile to Machine Language
Operating System

Major Responsibilities of the CLR

The runtime supplies the managed code with many services, such as cross-language integration, code access security, object lifetime management, and debugging support. Let's take a look at the more important responsibilities in more detail.

NOTE

The phrase "managed code" keeps coming up when discussing the CLR. Basically, code compiled with a language compiler that targets the Common Language Runtime is called managed code.

Safety and Security Checks

Before the CLR executes any code, that code is checked to make sure that it is type-safe in regards to memory locations. Each type has its own area of memory, and can't access the private data from other types. This means that code allowed to run in the CLR can only access memory locations that it is authorized to access. This prevents some of the crashes and tampering that can occur in other languages.

Code can also have specific security requirements that tell the CLR to verify that any client requesting the code in question meet those requirements. The security information resides inside the code executable files (DLLs or EXEs). Depending on the current security policy, certain operations may not be allowed as well.

Class Loading

When you start a .NET application, the Windows operating system recognizes that the executable was compiled for the CLR and passes control over to the CLR. The .NET Framework SDK install takes care of updating the operating system to be able to accomplish this. The CLR must find the entry point (*Main()* method) of the application and locate the class that contains that entry point.

The CLR is responsible for finding the proper version of the executable file and if it passes the security checks, loads the class in question, and activates the desired object when required. To find the proper version the CLR reads the information saved with the executable and searches a defined path to locate the desired file. Once found, the class loader can load the class in memory and cache the classes' type information so that it can just pull the information from the cache the next time the class is called. This is why the first time you run an application might seem a little bit slower than subsequent runs.

Object Lifetime Management

One of the historic hassles with development in the lower-level object-oriented languages is making sure that your objects are cleaned up when you no longer use them. Because the CLR handles that for you now, it's no longer a concern. The chance of memory leaks occurring and taking down a server is greatly reduced, and a big troubleshooting headache has been removed.

We discussed earlier how the CLR takes care of loading and activating objects. The CLR is also responsible for monitoring or tracing the operation of the object and removing the object when it is no longer referenced. Once an object is no longer referenced in any way from any other objects or variables, garbage collection will occur and return that memory for use.

NOTE

Garbage collection occurs based on many different calculations and can't be forced to occur. Therefore use of code intended to run when an object is destroyed (destructors) is not recommended. There is a *Dispose()* method that disables the object but leaves it in memory. This is where you would put in your cleanup code if it is still necessary.

Just In Time (JIT) Compilation

When you compile your code using Visual Studio .NET or one of the command-line compilers, that code isn't converted directly into machine language. It's converted to code called the Microsoft Intermediate Language (MSIL) or Common Intermediate Language (CIL). All .NET language compilers use this same process. This is how one .NET language can utilize objects created with other .NET languages. Both are compiled to the MSIL. To speed up the process of compiling the MSIL to native code for the associated CPU, the .NET Framework includes JIT compilers—one for each CPU architecture that .NET supports.

Each method for which MSIL has been generated is JIT-compiled when it is called for the first time, cached, then executed. The next time the method is executed, the existing JIT-compiled native code from the cache is executed. This process of JIT compiling and then executing the code is repeated until the application execution is finished. As mentioned earlier, the type safety is also checked at this time. As part of the compiling process, the code goes through the security checks noted earlier in the "Safety and Security Checks" section.

Cross-Language Interoperability

All languages targeted for the CLR must follow the Common Type System (CTS) rules for how their types are created and used. In addition, type information storage is the same across languages. So when the CLR looks up the type information to provide the proper services and execute the code, it doesn't matter what language it was written in, it's MSIL at this time. The MSIL is JIT-compiled and executed in the same way regardless of language. The result of this interoperability is that you get the following by using managed code:

- Inheriting from types created in other languages.
- Passing objects created in one language into a method of a class created by a different language.
- Common debugger across languages. You can step through code from one language to another as you debug.
- Error (exception) handling is the same across languages. An exception "thrown" in one language can be "caught" in another, and handled.

NOTE

One caveat with cross-language interoperability: In certain cases, one language supports functionality that another may not support. The example usually presented to describe this is if one language has support for unsigned integers (type *UInt32*) and you try to add a parameter of *UInt32* to your code in a language that doesn't support *UInt32*, your code wouldn't run. Keep that in mind when you have different languages involved.

Structured Exception Handling

Structured exception handling is a very significant part of the .NET Framework. The "structure" part is a control structure identical to that found in Java. It is composed of the *Try–Catch–Finally* block. This is similar to how C++ handles exceptions. We take another look at exception handling in the section "Structured Exception Handling Revisited" a little later on in this chapter.

Assemblies

Assemblies are the executables of the .NET Framework (DLL or EXE). Your source code project compiles into an assembly. You can choose to have more than one compiled file in your assembly. Assemblies are known as the smallest deployable unit. All managed types and resources are marked either as accessible only within their implementation unit or as exported for use by code outside that unit. In the runtime, the assembly establishes the name scope for resolving requests and the visibility boundaries are enforced. The runtime can find the proper assembly and the proper type when it needs to load that type to prepare for execution. All types are contained in assemblies.

MetaData

When your code is compiled for the CLR, more is going on that just compiling code. During compilation, the .NET-targeted compilers also produce the metadata that describes all of the types contained in the executables (EXEs or DLLs). This metadata includes type information that you would typically find in a COM-type library, and version dependency information. This version information is what the CLR uses to allow multiple executables with the same name residing on the same machine (in different directories). In addition, information describing all of the resources that your components use is also contained there. So the components are seen as "self-describing." When you start an application, the CLR looks to the assembly before loading to make sure that everything you need to execute that code is available and proper. The tracking down of dependency/versioning problems for days like you had to do with COM is now a thing of the past.

The information found in the metadata describes every element managed by the runtime, and any other information that is required by the runtime. Some examples are type, debugging, garbage collection, and versioning information. If you want to add more information than is provided by the compilers, you can create custom metadata that allows you to add attribute information to your assemblies that can be read and acted upon at runtime. The information found in the metadata includes the following:

- Assembly description:
  - Identity (name, version, culture, public key)
  - Other assemblies that this assembly depends on
  - Any security permissions needed to run your code
- Description of types (classes):
- Name, visibility, base class, and interfaces implemented
- Members (methods, fields, properties, delegates, events, inner types)
- Attributes:
  - Additional descriptive elements that modify types and members
  - Any custom attributes that you create yourself

Enhanced Deployment and Versioning Support

Deployment is one big area where the .NET Framework makes your life easier. You have the ability now to create *deployment projects*. These projects created within Visual Studio .NET use the Windows Installer technology. Deployment is totally different from previous versions of Visual Studio, so here we cover some of the highlights. You have various options to bundle your files for deployment:

- Use your assemblies as-is.
- Place files in a compressed CAB file. A good choice for large projects that would take a long time to download if installed over the Web.
- Create a Windows Installer package to take advantage of that technology. This gives you the most options as you are creating a Visual Studio .NET deployment project.
- Using a third-party installer.

You also have various options to distribute your bundled files:

- XCOPY or FTP. If you package your assemblies as-is, you can just copy the files over to an appropriate directory (using the proper namespace scope), and the application will run right from there.

- Have users download the code from a Web site and run your code from there.
- If you use Windows Installer to create your package, the installer can move the assemblies to wherever you need them if the security policy is met. You should understand these security rules prior to creating your install package.

Managed versus Unmanaged Code

Managed code runs in collaboration with the CLR during runtime. Managed code carries with it the metadata necessary for the runtime to provide services such as memory management, cross-language interoperability, code access security, and control of the object from instantiation through disposal. All code based on MSIL executes as managed code.

So basically unmanaged code is code created outside of the .NET Framework. This code also runs outside of the .NET Framework in the Microsoft Windows environment. The CLR has the ability to run unmanaged and managed code together, but there is a performance hit. Some common examples of unmanaged code are C++ classes compiled with a non-CLR compiler, COM components, and the classes that make up the Win32 API.

Interoperability with Unmanaged Code

The CLR takes care of interoperability between managed and unmanaged code, so it is quite seamless. This makes the decision of how and when to migrate your legacy code to .NET an easier one. You don't have to rewrite everything right away. The .NET Framework enables interoperability with COM+ services, the Windows operating systems, and COM components.

Namespaces

Namespaces provide a logical naming scheme for grouping related types, similar to Java packages. For example, the .NET Framework uses a hierarchical naming scheme for grouping types into logical categories of related functionality, such as Data Access, GUI, Reporting, and so on. Design tools can use namespaces to make it easier for developers to browse and reference types in their code. In the .NET Framework, a namespace is a logical design-time naming convenience, whereas an assembly establishes the name scope for types at runtime. Namespaces also aid the CLR in selecting the correct assembly to load when multiple assemblies have the same name.

For example, when you want to access a class found in the .NET Framework you would use the fully qualified name, which includes the namespace. In the reference *System.Data.Dataset*, *System.Data* is the namespace and *Dataset* is the type name. Microsoft states that all namespaces shipped by Microsoft will start with either *System* or *Microsoft*.

Because namespaces provide the scope for your types, the standard is to group types with related functionality in separate namespaces. *System.Data*, *System.Xml*, *System.Collections*, *System.Web*, *System.Net*, *System.Threading*, and *System.Security* are good examples of creating namespaces with logical groupings. These are found in the Base Class Library that Microsoft provides as part of the .NET Framework. We look into the Base Class Library and it's namespaces in the next section.

Developing Applications with the .NET Framework

To bridge the gap between the .NET Framework theory and the next chapter where you'll get right into ASP.NET, we cover some more programming-related information in this section. We take a look at the following:

- Development platforms
- .NET language choices
- Compilers
- Tools to make your .NET programming life easier
- Base Class Libraries

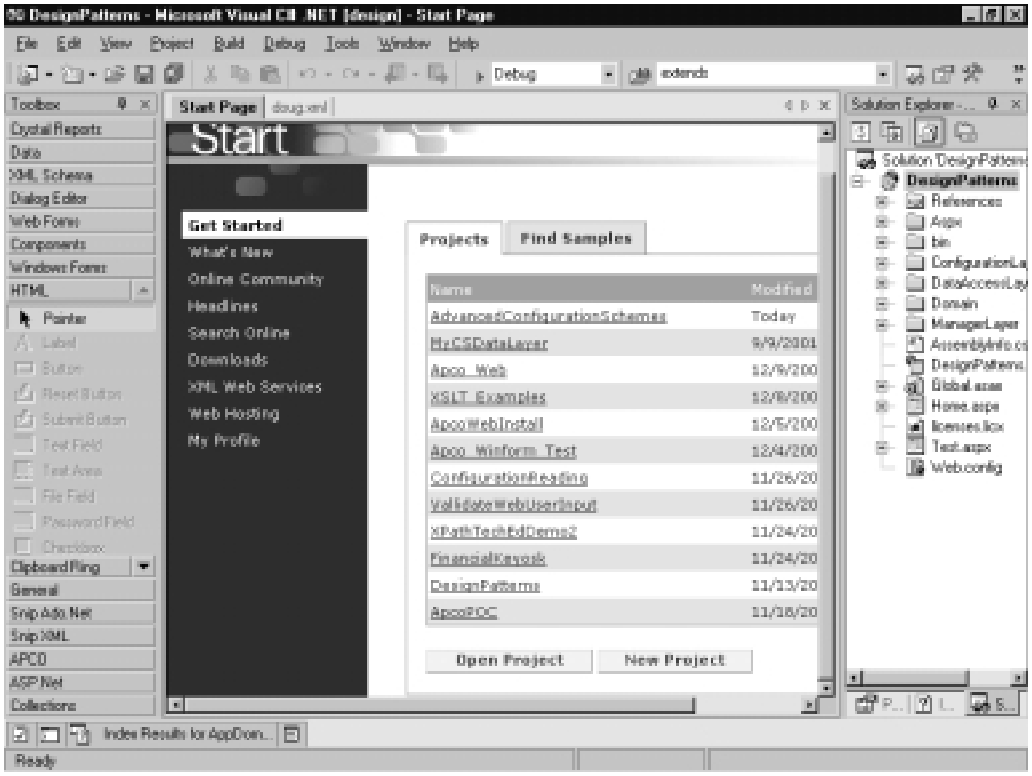
Development Platforms for .NET

You can create all of your code using a text editor such as Notepad if you like. The SDK provides all of the tools necessary to accomplish development at that level. You would need to learn all of the specific compiler options and debugger tools and probably take quite a while before you could be productive.

Third-party and open source tools are cropping up that can color-code keywords and provide some level of syntax checking. Some you can also link to a compiler so that you can code and compile in the same integrated development environment.

By far the best tool for .NET development we've used is Visual Studio .NET. The Professional Edition is in RTM (release to manufacture) condition already, so by the time you read this it should be widely available. This is the most productive development tool we've ever seen. You can code, compile, and build deployment projects all within the same development environment. Microsoft really focused on how to make developers more productive and put that research in this tool. Figures 2.5 shows a typical project open in Visual Studio .NET Enterprise Architect Edition.

Figure 2.5 Visual Studio .NET Integrated Development Environment

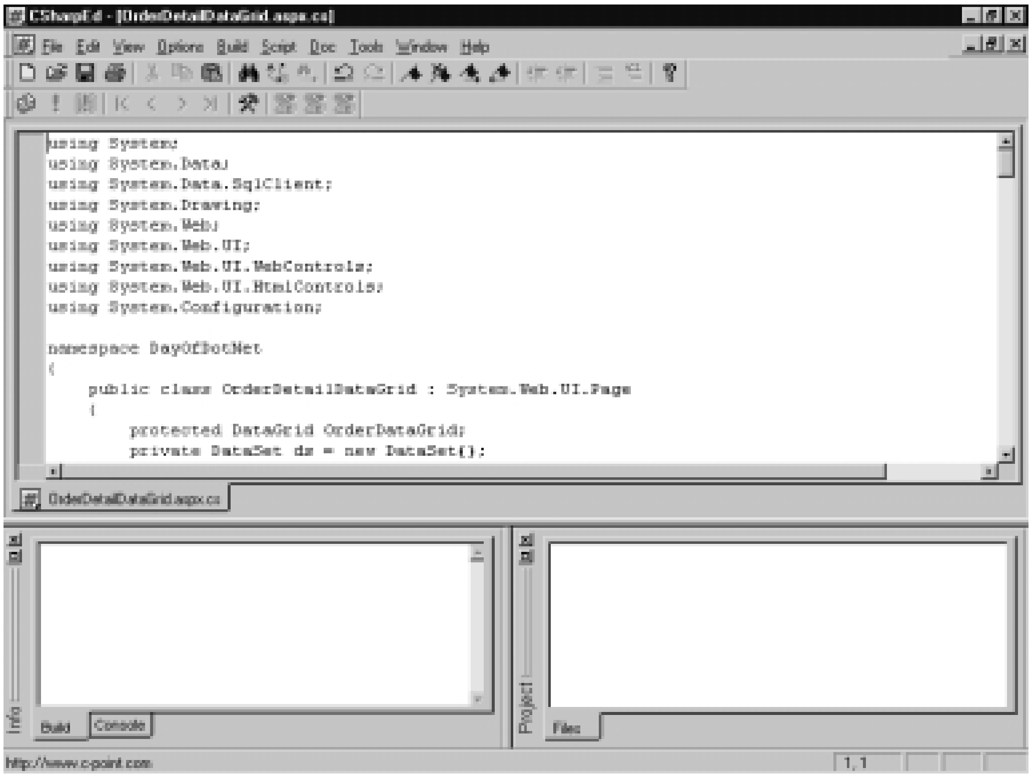


Name	Modified
AdvancedConfigurationSchemas	Today
MyCSDataLayer	9/9/2001
Apco_Web	12/9/200
XSLT_Examples	12/8/200
ApcoWebInstall	12/5/200
Apco_Winform_Test	12/4/200
ConfigurationReading	11/26/20
ValidateWebUserInput	11/26/20
XPathTechEdDemo2	11/24/20
FinancialKayak	11/24/20
DesignPatterns	11/13/20
ApcoPOC	11/18/20

Figure 2.6 depicts another C# editor available on the Web. More and more editors of various levels of cost and functionality are showing up on the Web every day.

Once you have your development platform selected, you need to consider other issues as well. We look at a few of those next.

Figure 2.6 Antechinus C# Editor


CSharpEd - [OrderDetailDataGrid.aspx.cs]
File Edit View Options Build Script Doc Tools Window Help
OrderDetailDataGrid.aspx.cs
Info
Build
Console
Project
Files
http://www.c-point.com
1, 1

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
using System.Configuration;

namespace DayOfDotNet
{
    public class OrderDetailDataGrid : System.Web.UI.Page
    {
        protected DataGrid OrderDataGrid;
        private DataSet ds = new DataSet();
```

Language Choice

Currently, many languages are being targeted for the .NET Framework. Visual Studio .NET Beta 2 includes VB.NET, C#, managed C++, and Jscript. J# (allows using Java syntax for building .NET applications) is now in beta and downloadable from the MSDN site, and it also integrates with Visual Studio .NET. And currently, projects are underway for Cobol.NET, Fortran.NET, Eiffel.NET, and more.

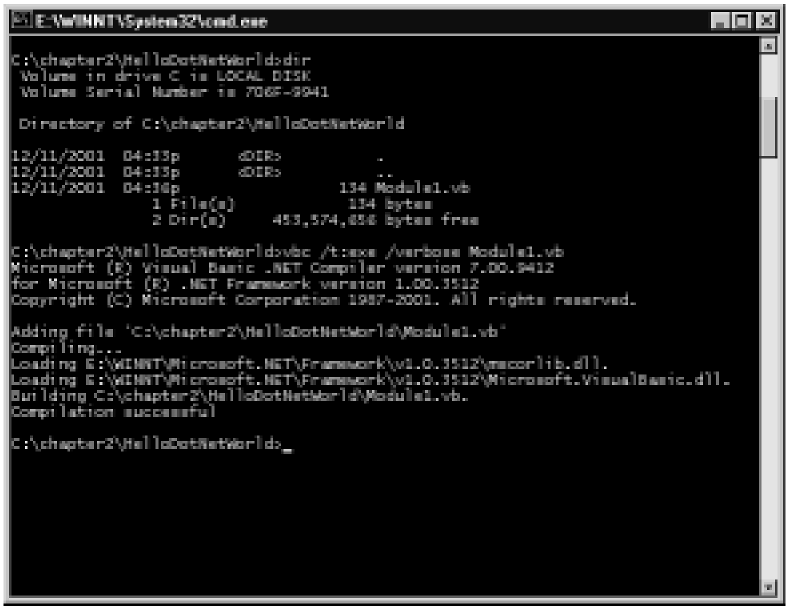
Because all .NET language compilers create Microsoft Intermediate Language code, there is theoretically no difference within the .NET Framework as far as performance goes. So the compelling performance or ease of coding reasons for developing in a specific language has gone away. You can basically choose a language that you are comfortable with and not lose major benefits from one language to another. Some say that choice of language is a lifestyle choice. The target we've chosen for this book is VB.NET.

Using the Compilers

As mentioned earlier, you can build .NET applications using Notepad and the command-line compilers if you want to. The compilers that ship with the SDK are csc.exe (C#), vbc.exe (VB.NET), and jsc.exe (Jscript .NET). Figure 2.7 depicts a subdirectory with one file called Module1.vb. The command *vbc /t:exe /verbose Module1.vb* translates to the following:

- *vbc* = Run the vbc.exe compiler with the following options.
- */t:exe* = The "target" for the compilation is an exe file.
- */verbose* = Display all of the output information.
- *Module1.vb* = The source file that we want to compile.

Figure 2.7 Example of Command-Line Compilation Process



```
C:\chapter2\HelloDotNetWorld>dir
 Volume in drive C is LOCAL DISK
 Volume Serial Number is 706F-9941

 Directory of C:\chapter2\HelloDotNetWorld

12/11/2001  04:33p      <DIR>          .
12/11/2001  04:33p      <DIR>          ..
12/11/2001  04:36p                 134 Module1.vb
               1 File(s)            134 bytes
               2 Dir(s)     453,574,656 bytes free

C:\chapter2\HelloDotNetWorld>vbc /t:exe /verbose Module1.vb
Microsoft (R) Visual Basic .NET Compiler version 7.00.9412
for Microsoft (R) .NET Framework version 1.00.3512
Copyright (C) Microsoft Corporation 1987-2001. All rights reserved.

Adding file 'C:\chapter2\HelloDotNetWorld\Module1.vb'
Compiling...
Loading E:\WINNT\Microsoft.NET\Framework\v1.0.3512\mscorlib.dll.
Loading E:\WINNT\Microsoft.NET\Framework\v1.0.3512\Microsoft.VisualBasic.dll.
Building C:\chapter2\HelloDotNetWorld\Module1.vb.
Compilation successful

C:\chapter2\HelloDotNetWorld>_
```

For C++ managed code, you would compile your C++ code with the */clr* compiler option. Please note that there are many restrictions to using the */clr* option, so review the documentation and understand what you are doing before trying to use managed extensions.

Running the compilers from the command line gives you the same effect as setting project properties or compiler options in an integrated development environment like Visual Studio .NET. The example in Figure 2.7 is about as easy as it gets. Using Visual Studio .NET, or other integrated development environments for .NET development, sure manages that process a lot easier for you.

Tools

A variety of tools are provided with the .NET Framework SDK that aid developers in building better .NET applications. We cover a few of the more important ones here:

- ILDasm
- NGEN
- Debugging tools

ILDASM.exe

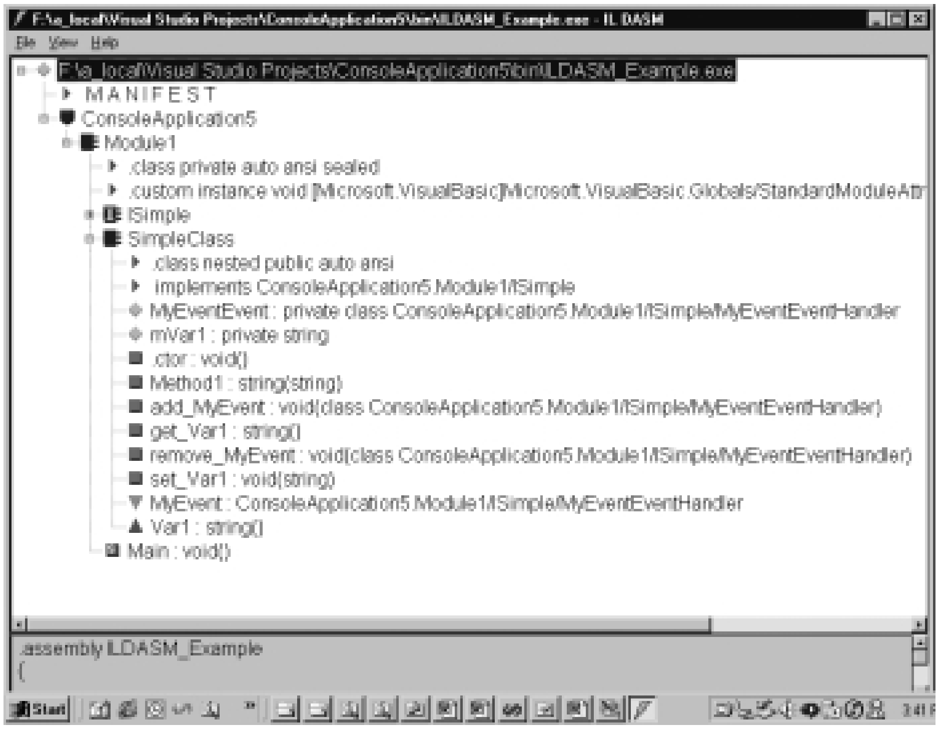
ILDasm (Intermediate Language Disassembler) is a tool that .NET developers need to have at their fingertips. Similar to JavaJava classpath problems, a great deal of .NET problems with the compiler not being able to find classes is due to name space issues. This tool allows you to look at an assembly and see the metadata for every element in the assembly. Double-clicking on an element actually brings up the MSIL so that you can see exactly what the compiler did to your code. So if you receive an exception concerning types that can't be found by the compiler, looking at the metadata for the assembly quickly allows you to locate the type in question and where it is located. Figure 2.8 shows an example of what ILDasm can show. Double-clicking on an element shows the intermediate language code for that element so that you can see the code created by the compilers.

NGEN

The place where the machine-specific assembly code is located is called the Native Image Cache. You can see this in the \winnt\assembly directory. Each assembly has a directory for the assembly, and a corresponding native image directory. When the CLR needs to run code from an assembly, it first looks in this cache to see if it can avoid having to recompile the assembly to machine code and then run.

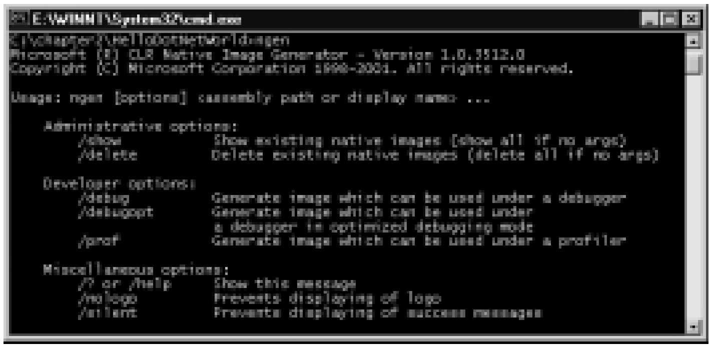
What NGEN.exe provides is the compilation to native image code up front. So you would probably want to do this on your production machine when a new build is ready for deployment. Obviously, any change and redeployment of the assembly would require a different version to be deployed, or run another NGEN compilation so that the existing outdated native image code won't be run. This is because the CLR always looks in the native directory first. Figure 2.9 shows the various command-line switches available for NGEN.

Figure 2.8 ILDasm.exe View of a .NET Executable (DLL or EXE)


F:\a_local\Visual Studio Projects\ConsoleApplication5\bin\ILDASM_Example.exe - IL DASM
File View Help
Start

```
F:\a_local\Visual Studio Projects\ConsoleApplication5\bin\ILDASM_Example.exe
  MANIFEST
  ConsoleApplication5
    Module1
      .class private auto ansi sealed
      .custom instance void [Microsoft.VisualBasic]Microsoft.VisualBasic.Globals/StandardModuleAttr
      ISimple
      SimpleClass
        .class nested public auto ansi
        implements ConsoleApplication5.Module1/ISimple
        MyEventEvent : private class ConsoleApplication5.Module1/ISimple/MyEventEventHandler
        mVar1 : private string
        .ctor : void()
        Method1 : string(string)
        add_MyEvent : void(class ConsoleApplication5.Module1/ISimple/MyEventEventHandler)
        get_Var1 : string()
        remove_MyEvent : void(class ConsoleApplication5.Module1/ISimple/MyEventEventHandler)
        set_Var1 : void(string)
        MyEvent : ConsoleApplication5.Module1/ISimple/MyEventEventHandler
        Var1 : string()
      Main : void()

.assembly ILDASM_Example
{
```

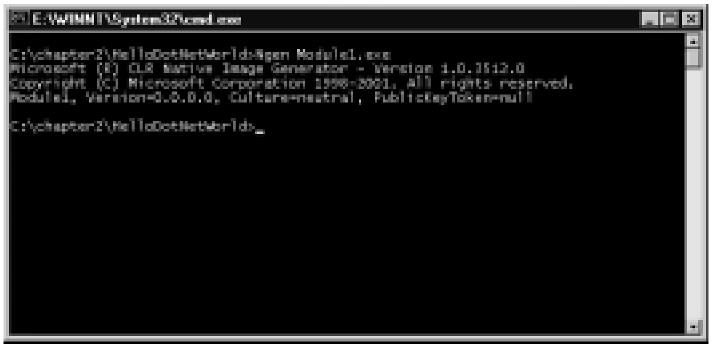
Figure 2.9 NGEN.exe Options



```
C:\chapter2\HelloDotNetWorld>ngen
Microsoft (R) CLR Native Image Generator - Version 1.0.3512.0
Copyright (C) Microsoft Corporation 1998-2001. All rights reserved.

Usage: ngen [options] <assembly path or display name> ...

    Administrative options:
        /show           Show existing native images (show all if no args)
        /delete         Delete existing native images (delete all if no args)

    Developer options:
        /debug          Generate image which can be used under a debugger
        /debugopt       Generate image which can be used under
                        a debugger in optimized debugging mode
        /prof           Generate image which can be used under a profiler

    Miscellaneous options:
        /? or /help     Show this message
        /nologo         Prevents displaying of logo
        /silent         Prevents displaying of success messages
```

Figure 2.10 shows the output of a successful compilation. Note that the output displays all of the versioning information for the assembly that you ran NGEN on. Assuming that you would use NGEN for code that you're putting into production, displaying the full version information for assembly gives you a chance to double-check that the proper assembly is being compiled into machine code.

Figure 2.10 Successful NGEN.exe Compilation

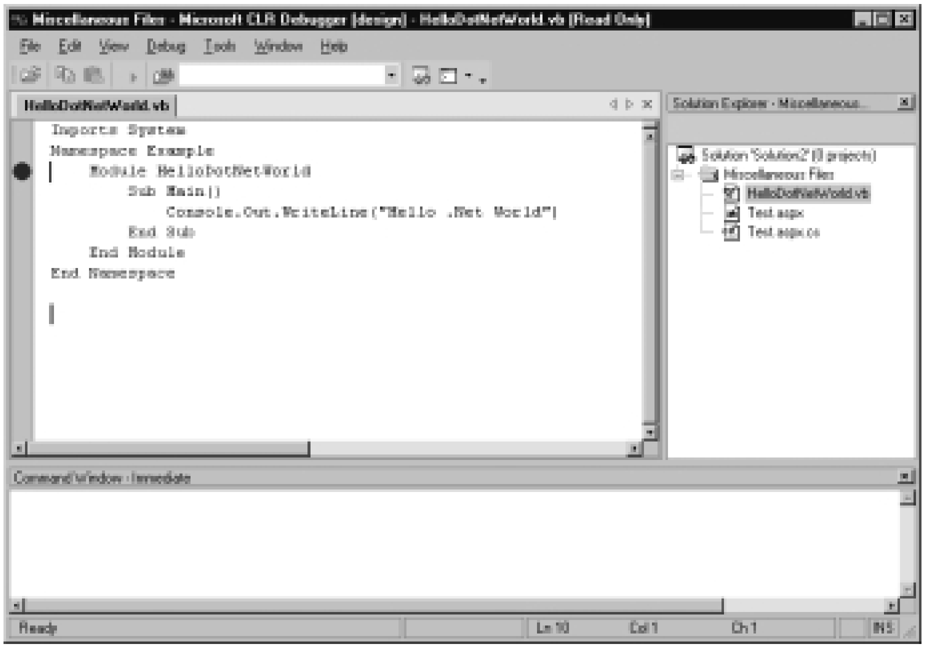


```
C:\chapter2\HelloDotNetWorld>Ngen Module1.exe
Microsoft (R) CLR Native Image Generator - Version 1.0.3512.0
Copyright (C) Microsoft Corporation 1998-2001. All rights reserved.
Module1, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null

C:\chapter2\HelloDotNetWorld>_
```

Debugging Tools

The two debuggers that are included with the .NET Framework SDK download are CorDbg and DbgClr. CorDbg is a command-line debugger. DbgClr is a Windows GUI debugger. Figure 2.11 is an example showing DbgClr. It looks very similar to the Visual Studio .NET integrated development environment.

Figure 2.11 DbgClr.exe GUI



Miscellaneous Files - Microsoft CLR Debugger [design] - HelloDotNetWorld.vb [Read Only]
File Edit View Debug Tools Window Help
HelloDotNetWorld.vb
Solution Explorer - Miscellaneous...
Solution 'Solution2' (0 projects)
Miscellaneous Files
HelloDotNetWorld.vb
Test.aspx
Test.aspx.cs
Command Window - Immediate
Ready
Ln 10
Col 1
Ch 1
INS

```
Imports System
Namespace Example
    Module HelloDotNetWorld
        Sub Main()
            Console.Out.WriteLine("Hello .Net World")
        End Sub
    End Module
End Namespace
```

In both tools, you can perform cross-language debugging. You can also attach and debug running processes. For both tools, you must have already compiled the code in question with the */debug* switch. These are tools that you need to take out and play with to get the most out of them.

Base Class Libraries

In order to use the services that the .NET Framework provides, the Framework exposes a number of base classes that are easily referenced and used. Some of the more important ones are shown in Table 2.1 to give you an idea of how the base classes are packaged.

The .NET Framework is "strongly typed." This means that everything is an object. And the objects follow a hierarchy starting from *System.Object*. If you don't explicitly code your class to inherit from another class, it is implied that your class inherits from *System.Object*. The *System* namespace is the "root" namespace for Base Class Library. Everything is an object in the .NET Framework, and everything ultimately inherits from the *System.Object* type. Because it's a given that if no explicit inheritance is defined in your class, your class inherits from *System.Object*. All of the base data types are found in this namespace as well (like *String*, *Byte*, *Array*, and so on). Some of the most basic services that the .NET Framework provides are handled by classes in the *System* namespace, such as exception handling, garbage collection, and the system environment. Take a look at Table 2.1 to give you an idea of the breadth of functionality provided by the Base Class Library classes.

Table 2.1 Some Important Base Class Library Namespaces

Namespace	Purpose
System.Data	Namespace that is made up of the ADO.NET classes. Provides data access services for any type of application built for the .NET Framework.
System.Collections	Provides different types of classes and interfaces to create collections of objects. Some examples are *Arraylist*, *Hashtable*, *Dictionary*, *Stack*, *SortedList*, and *Queue*. From the base collection types, you can create type specific collections very easily.

Continued

Table 2.1 Continued

Namespace	Purpose
System.Configuration	Allows access to the information in the configuration files that you store with your applications. If the default "handlers" (configuration file readers) don't meet your needs, you can create your own. All of the configuration files in the .NET Framework are in XML format.
System.Diagnostics	Used for debugging your code and allows for code tracing. The tracing functionality provided in ASP.NET alone would cause me to switch from ASP development to ASP.NET development. What a timesaver! Also included are classes that allow starting operating system processes, monitor performance of the system, and write information from the application to the operating system's event log.
System.IO	File and directory classes provide information and operations. Reading/writing files and creating/using data streams.
System.Net	Provides access to different network protocols. Most notable are the *WebRequest* and *WebResponse* classes. The classes in this namespace wrap functionality for network and Internet communication to enable you to worry more about solving your business problem and not about the nitty-gritty of specific protocols.
System.Reflection	A very powerful group of classes and interfaces that allow runtime access to the metadata for your assemblies. With this information, you could create objects and run methods on those objects at runtime which gives you an enormous amount of flexibility in your applications. You can also use reflection to view any custom information that you created and stored within your assemblies.

Continued

Table 2.1 Continued

Namespace	Purpose
System.Runtime.InteropServices,	Classes provide functionality for accessing COM objects and the Windows API components from the Common Language Runtime. These classes can wrap COM components with .NET classes and add functionality into .NET classes so that calls can be made to COM components. Either way it's a performance hit to do this.
System.Runtime.Remoting	The *System.Runtime.Remoting* namespace provides for the creation and configuration of distributed applications. This gives you the ability reference objects on remote machines, similar in some ways to what DCOM provides.
System.Text	*System.Text* contains classes for string manipulation and formatting, and character set encoding/decoding. You can really improve performance if you manage your strings well.
System.Web, *System.Windows.Forms*	These namespaces are covered in the applicable ASP.NET and Windows Forms sections later in the chapter.
System.Threading	Provides classes and interfaces that enable multithreaded programming. Thread pooling is included with classes for managing your threads. This is a big change for VB6 developers used to the single-threaded apartment (STA) threading model.
System.Xml	Provides the functionality to create and read standard XML documents. Support for XML schemas, soap, and XSL/T transformations are included. An XML query language called XPath is also found in this namespace.

This is just a fraction of the Base Class Library. You would be well served by going through the documentation provided with the SDK, and step through the samples provided with the SDK. It's a great starting point to learning what functionality you have provided right out of the box. To use a Base Class Library, you can use the full namespace path or the short way. Figure 2.12 is a simple example of using a Base Class Library namespace to take advantage of its functionality using VB.NET.

Figure 2.12 Using a Namespace from the Base Class Library

```
Imports System
Imports System.Collections

Module MyListModule1

    Sub Main()
       Dim myList As New ArrayList()

       myList.Add("ItemOne")
       myList.Add("ItemTwo")

      Console.Out.WriteLine("The first item is " & _
          myList.Item(0))

    End Sub

End Module
```

In this example, the *Imports* statement is where you tell the compiler that you are referencing the *System* namespace. That allows you to use short notation to reference the associated namespace. If you didn't want to use the *Imports* statement, then to use the *Console* class you would be required to include the full namespace notation (*System.Console.Out.Writeline()*).

Structured Exception Handling Revisited

Structured exception handling gives you the ability to anticipate what errors could occur, and enclose the code that may cause those particular errors. If an error is "caught" in your monitored code, you can invoke a predefined handler than can respond in a manner that can keep you up and running.

The .NET Framework provides its own exceptions for exceptions such as *FileNotFoundException*, *IndexOutOfBoundsException*, and *SecurityException*. You have the ability to create very complex error-handling schemes by inheriting from the *System.Exception* and associated exception classes to create user-defined errors and a common way of handling all exceptions in your application.

You can also grab the stack information at the time of the exception and pass that back up through the layers to where you want to evaluate and/or save that information for troubleshooting purposes. Stack trace information contains most of the information located on the stack at the time of the exception, so you can see what methods were called and exactly where the error occurred. Usually you get the method, filename, and the line of code where the exception occurred. Figure 2.13 shows an example of using the *Try–Catch–Finally* block. The descriptions for the lines of code are listed in Table 2.2.

Figure 2.13 Structured Exception Handling

```
Imports System
10 Imports System.IO
20 Public Class FileExceptionExample
30 Public Sub DoFileStuff(ByVal strFileName as String)
40  Try
50   Dim fs As New FileStream("c:\myfile.txt",
              FileMode.Open)
60        'File operations go here …..
70        'When file operations complete then close the stream
80        fs.Close()
90

100   Catch fnfException As FileNotFoundException
110     Throw New FileNotFoundException("Hey file isn't
        there", fnfException)
120   Catch eosException As EndOfStreamException
130       Throw New EndOfStreamException("put your message
          in here")
140   Catch e As Exception
150         Throw New Exception("put your message in here")
160
170   Finally 'optional
180
190   End Try
200
210   End Sub
```

Table 2.2 Line Number Descriptions for Figure 2.13

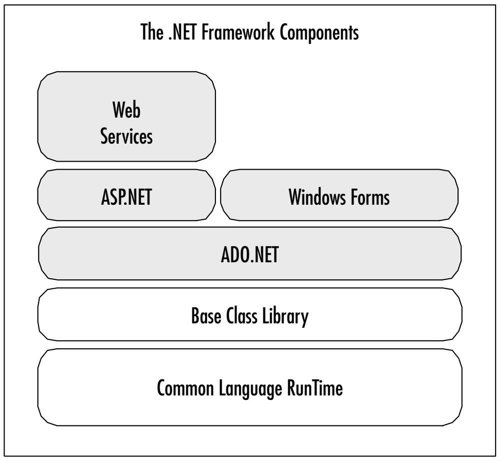
Line Numbers	Description
40–100	Performing a file input/output operation. From experience, you may know that several different exceptions could occur with accessing the file. With that in mind, place the *Try* statement above the code that could cause exceptions.
100	A likely problem could be that the passed-in string for the filename (*strFileName*) doesn't correspond to a file at the location specified. This is a problem that you would want to catch in the code, so you can place a *Catch* statement for the *FileNotFoundException*.
110	The *FileNotFoundException* is handled by passing to the method that called the *DoFileStuff* method a message and the exception object variable (*e*) so that it can handle the exception appropriately. The *throw* statement does that. You have a few different options for the *throw* method as far as what you return to the calling method. You can just as easily handle the exception here, on line 110 instead of throwing it, and continued on with operation. If you don't throw the exception, the code in the *Finally* block (line 170) will execute, and operation will continue with the code after the *Finally* block.
120	In this line, you're catching an *EndOfStreamException*. You can catch multiple exceptions for a given *Try* block. When an exception occurs, the CLR looks at the *Catch* statements from top to bottom to find the proper handler for the exception that occurred.
140	This *Catch* statement basically says, "If an exception occurs that I haven't anticipated and coded a *Catch* block for, I'll take care of it here."
170	The *Finally* statement is optional. This is where you would write "cleanup" code to release resources that may have been obtained before the exception. For example: When performing data operations, you open a data connection, then get an *SQLException* when you try to access the data. In this case, you would want to close that data connection.
190	The *End Try* is placed here to completely wrap the exception handling in the *Try* block. You can have multiple *Try* blocks in a method, and you can nest *Try* blocks.

Generally, in our opinion, you want to try to handle the exception as close to where it occurs as possible. Most large enterprise applications have a "global" error handler that does things like provide notifications to the user of the exception, logs the exception, and sends notifications via e-mail or pager to support personnel, all depending on the severity of the error.

Components in the .NET Framework

The .NET Framework's "layered" approach provides a firm foundation for building .NET applications. The CLR is at a higher level of abstraction than the Windows operating system. The Base Class Library provides a higher level of abstraction than the CLR. Now in Figure 2.14, you can see that the major components for building a .NET application like ASP.NET are at a higher level of abstraction still. Each layer makes development more productive, and still allows the ability to drill down into the lower levels to get at specific functionality. The different frameworks also provide excellent models for developing your own frameworks. And you can always subclass the existing functionality to create your own.

Figure 2.14 The .NET Framework with Major Components



ASP.NET

As shown in Figure 2.14, ASP.NET is a framework that sits architecturally on top of the .NET Framework. So it takes the base class libraries for Web development and abstracts them to a higher level to make creating Web applications even easier.

Web Form pages are the central display mechanism for ASP.NET applications. They are made up of .aspx files compiled with an associated .NET language class file that contains the code. This is known as *code-behind* and is the default when you create a new Web form in Visual Studio .NET. It is possible to put the code directly on the same page as the HTML, but we don't recommend it. The whole idea is to separate the display from the logic. The Web forms pages are compiled prior to execution and cached so performance overall should prove to be better than the interpreted Active Server Pages (ASP).

Performance is also improved by the new caching classes that enable smart caching. That gives you the ability to store information that doesn't change much but is expensive to obtain (like through database operations) in memory. It's called the cache, and when that information is needed, it's grabbed from memory, instead of having to perform the data operations. You can set up time durations for the cache and dependencies such as if file *x* changes, then empty the cache of *y* information. There is even a caching namespace, called *System.Web.Caching*, to give you very low-level control over what you can cache. Careful use of this feature can give you significant performance enhancement.

ASP.NET gives you server side controls which expose you to a whole new world of events and server side processing that didn't exist in traditional ASP. The page is created on the server, and pure HTML is rendered back to the client so they can be run on any browser. Visual Studio .NET is an awesome environment for creating ASP.NET Web applications. The VB-like ability to drag and drop controls and position them on the forms, double-clicking on the forms controls to call up the code and create events for that control, and the debugging make building Web applications really fun. It takes out the drudgery that was common with ASP development.

As you'll see later in this book, most of the benefits that Visual Studio .NET gives you with ASP.NET development are also available for mobile Web development. This includes mobile Web forms and controls.

ADO.NET

As was shown in Figure 2.14, ASP.NET has full access to the *System.Data* name space of ADO.NET. So ADO.NET provides data management services to all .NET Framework components. ADO.NET is similar to ADO, so the learning curve won't be that steep with ADO.NET. The additional object-oriented nature of ADO.NET is the major difference.

In Beta 2 of the .NET Framework SDK, only two data "managed providers" are included. The OleDb managed provider and SqlServer 7 and above managed provider. Data operations using the SQLServer provider provide much better performance because there is no middle layer of translation that you will find with the OleDB managed provider. The "middle layer" that we refer to is the OleDb/ ODBC layer that the OleDb Data Provider requires for operations. The SQLServer provider communicates via the tabular data stream (TDS) protocol, which is the native protocol for SQLServer. This optimization is what provides the major performance difference between the SQLServer and OleDb Data Providers.

DataReader classes are provided in ADO.NET that process a data stream in a read-only, forward-only manner. Currently there is an *SqlDataReader* for SQLServer, and *OleDbDataReader* for the other data providers. This provides the most efficient way to get read-only data for filling simple lists and tables on your Windows Forms and Web Forms.

DataSet is an in-memory cache of *Data-Tables* that allows you to get your data and work with it just like you would in a normal database. This includes multiple tables and relationships involved and maintained in *dataset*. You can tell the *dataset* to update the main datastore and refresh the *dataset* as often as you need to. You can create and manually populate *DataSets* and use them simply as a temporary database. The data in a *DataSet* can be persisted as XML which can be very useful when you need to transport that data over the Internet, or to applications running on different platforms like mainframe or Unix.

The *DataSet* usage scenario is focused on obtaining data from a data source, then disconnecting from that data source and managing how the disconnected data is used. Using disconnected data makes the most sense in a Web environment. One of the things you look for when developing applications for the Web is to make sure that if you are performing operations that carry a high resource overhead, such as setting and maintaining data connections, you try to get rid of those resources as soon as you're done with them. Having a data connection open just long enough to get a disconnected set of data allows the server to reclaim those resources sooner, so the server will be able to handle more users. If possible, having the data set on the client to be worked with and updated in batch, minimizes round trips to the server, which improves scalability as well. ADO.NET is covered in much more depth in Chapter 6.

VB.NET

VB.NET takes the impression of non-VB programmers that VB is a "toy" language, and dispels that once and for all. You still have the productivity that VB has always been famous for. Now you have the fully object-oriented language features and the ability to actually see what is going on behind the scenes. Nearly every feature that is available to the other .NET languages is open to VB.NET. All of this new power will require VB developers that don't have object-oriented programming experience to increase their knowledge in that area. Building scalable enterprise applications requires careful design. The good news is, if developers have been building object-oriented applications and components in VB using the Windows DNA architecture, the learning curve won't be too steep. And there are tools you can use and steps you can take to move your current applications closer to .NET so that your migration will be as painless as possible.

Because VB.NET is targeted at the .NET Framework, it carries with it all of the benefits and available services provided by the Framework. The .NET Framework also enables interoperability between objects you create with any .NET programming language. Some big language differences between VB.NET and previous versions are listed here:

- Structured exception handling.
- VB.NET is now a fully object-oriented language, including inheritance.
- VB.NET is fully integrated into the .NET Framework with access to the same services provided to C#, managed C++, and the growing number of .NET Framework–targeted languages.
- Delegates have been included. Delegates are type-safe, object-oriented function pointers. They do more than simple function pointers in C++, and are a big difference between .NET languages and Java. Delegates can handle more than one method. You can use delegates to identify event handlers and also use them with multithreaded applications.
- The threading model has been changed from single-threaded apartment to free threading.

An upgrade wizard is provided with Visual Studio .NET that will help you upgrade existing VB6 applications to VB.NET. The wizard also creates an upgrade report that documents the upgrade process and any errors that occurred during the process. That and the loads of documentation regarding upgrades should make the process as easy as possible. Figure 2.15 shows a simple example of writing "Hello .NET World" to the console window.

Figure 2.15 Hello .NET World in VB.NET

```
Imports System
Namespace Example
    Module HelloDotNetWorld
        Sub Main()
            Console.Out.WriteLine("Hello .NET World")
        End Sub
    End Module
End Namespace
```

NOTE

Visual Basic 6.0 used the single-threaded apartment threading model. In the STA model, a component's methods are tied to a single thread (apartment). Therefore, any other methods for that component that could run has to wait for the running method to finish). The .NET Framework uses a multithreaded apartment (MTA) threading model, which allows multiple threads to be available to an apartment. This adds to the responsiveness that the user experiences.

C#

C# is a language created from the ground up. It has taken the best from existing object-oriented languages, such as Java and C++, and improved on perceived shortcomings from those languages. The intent was to create a language that would give the productivity of VB and the power of C++. C# is centered on building components. Many comparisons have been made between Java and C#. In our opinion, C# has many similarities to Java and offers more.

Any Java or C++ programmer will have no problem at all reading and understanding the C# syntax. Like Java, it has taken the best of C++ and left out the areas that can cause the most headaches, such as function pointers and the ability to step on unauthorized memory locations that caused programs to crash in bad ways.

Figure 2.16 shows the same simple program we saw in Figure 2.15. Note the minor syntax changes between the C# example and the VB.NET examples.

Figure 2.16 Hello .NET World in C#

```
using System;
namespace HelloDotNetWorld
{
   public class HelloDotNetWorld
   {
      public static void Main()
      {
        Console.Out.WriteLine("Hello .NET World!");
   }
      }
}
```

In most cases the difference between C# and VB.NET is in the syntax. So again, choice of language is really up to developer preference. The expectation is that C++ and Java developers will gravitate to C#, and VB developers will gravitate towards VB.NET. Time will tell.

Windows Forms

The .NET Framework has two primary user interfaces, ASP.NET Web Forms, and Windows Forms. Windows Forms is a framework that provides the new platform for Microsoft Windows application development, based on the .NET Framework. This framework provides a clear, object-oriented, extensible set of classes that enable you to develop what is known as rich Windows applications. They are applications that allow you to take advantage of what the Windows operating system provides for creating user interfaces.

The *System.Windows.Forms* namespace provides the classes that enable you to create rich client applications. Because everything in the .NET Framework is a type, you can reuse and extend your forms classes by inheriting from them and then making whatever additions or changes you may need to.

The *Form* class is simply a window-like container for the controls that you place on the form. You have the ability to derive from the *System.Windows.Forms.Form* class to create new forms or use an existing form as a template and derive from the pre-existing form. Creating form "templates" could save a lot of work for larger applications. Any changes to the base *Form* class are automatically available to the other *Form* classes that inherit from that base class.

As with the *Form* class, you can use an existing control or inherit from an existing control to create your own. Windows Forms controls are called *rich* controls because they carry with them the rich Windows user interface and formatting. In comparison, controls in ASP.NET Web Forms are limited to what HTML 3.2 Web browsers can provide as far as user interface.

Control layout has been enhanced with the Windows Forms framework. If your forms can be resized at runtime, and you want the controls to resize with the form, you would use the *Dock* property. This sets the controls position within the form and maintains that during resizing.

If you require that the control maintain a certain distance between the control and it's container upon a resizing of the form, you would use the *Anchor* property. You can anchor a control to one or more sides of its container so that the "anchor" position is maintained with respect to the chosen sides of the container.

Anchor and *Dock* properties are also available at runtime. You have an unlimited number of different combinations of containers/controls/control properties so that you can set up your forms in any way you like.

With Visual Studio .NET, you drag and drop the rich set of Windows controls onto the Form design interface just like in previous versions of Visual Basic

Web Services

Web services are the latest and greatest "new thing" to be hyped in the industry. Microsoft, IBM, Sun, and other companies are getting on the Web services bandwagon. Web services are basically components that can be referenced and their methods run via standard Web protocols, such as SOAP and XML. The industry is moving towards standard ways of describing the functionality that a Web service provides, and how a Web service is used. Because standard Web protocols are used, it shouldn't matter what language the Web service is written in or what platform the Web service is running on. As long as you communicate over HTTP with XML and are using the SOAP, you should be able to send information to and receive information from a Web service. Because the communication is over HTTP, it can make it through firewalls without any problem, but the ability to encrypt the communication and/or use Secure Sockets Layer is also available to provide the same level of security as ASP.NET Web applications.

As long as a client application has Internet access, it can utilize a Web service. This includes interaction with other Web services, ASP.NET Web applications, and rich Windows Forms applications. The vision of the next incarnation of the Internet is made up of applications utilizing functionality from Web services all over the Internet. And users pay for the service every time they access the Web service. This means that a company that provides a particular set of functionality can now open that functionality to the world on the Internet—a totally new revenue stream is the vision.

For developers using Visual Studio .NET, accessing a Web service for its functionality is as easy as adding a reference to the service and then coding to that reference just like it was a component on your machine. Creating a Web service is almost as simple. To create a Web service with publicly available Web methods, you simply create a Web service project, which provides a boiler plate class, then put <*WebMethod*> attributes above each public method's signature.

Web services are integrated with the ASP.NET framework, so they have the same set of services available to them as ASP.NET applications have. Some examples are data caching, ADO.NET, session state, and security.

Summary

We've covered a lot in this chapter. We started with obtaining and installing the .NET Framework SDK. The system requirements can cause some confusion. Just be careful that you meet them so that you can save yourself a lot of headaches. The .NET SDK install is very easy. The thing that does catch some people is the requirement for ASP.NET that IIS be installed first.

We went over the Common Language Runtime, the "sandbox" for .NET program execution. The CLR manages the code you write from verifying access, finding the proper assembly, through code execution. The code you write and compile for the CLR is converted to Microsoft Intermediate Language (MSIL) and is called managed code. When the CLR needs the code executed it just-in-time–compiles it then runs it. When the objects are no longer referenced, garbage collection will take care of them as appropriate.

Because all .NET languages compile to MSIL, you have language interoperability between .NET languages for inheritance, referencing objects, error handling, and debugging. Interoperability with unmanaged code is also possible with the CLR turning over control of that code to the Windows operating system.

Deployment is much less of a hassle with versioning. Versioning defines an executable in different ways (name, version, culture, public key). This allows multiple versions of the same file to be deployed on the same machine, with no conflicts. You can bundle your files for deployment in different ways, from XCOPY to Windows Installer packages and CAB files.

Metadata makes the executables self-describing which opens up many possibilities for dynamic loading of objects and method calls at runtime. The metadata contains information for every element managed by the runtime along with the MSIL. This makes an assembly a complete deployable unit. Therefore, you have no need for the Windows registry.

Namespaces allow you to create scope for your common groups of functionality, and provide another way to disambiguate one assembly from another with the same name. The Base Class Library provides excellent examples of how to intelligently use namespaces.

In the "programming" section, we discussed some more important concepts that should help you get a jump start on your development. Once you become familiar with the Base Class Library, you will be well on your way to "becoming one" with the .NET Framework. ILDasm.exe and the debugging tools will make your development time more efficient and more pleasurable. NGEN.exe compiles your code to the native machine codes so the CLR will bypass the just-in-time–compilation and run the machine code directly, significantly improving performance.

The last section was on the .NET Framework's major components (ASP.NET, ADO.NET, Windows Forms, and Web services). Most of these are actually frameworks in themselves put there to make your life easier. You can use the frameworks provided, or create your own by using the Base Class Library directly. But so far we've been able to get by with very little inheriting from the ASP.NET or the Windows Forms frameworks. Microsoft has really spent a lot of time and money researching developer productivity issues, and the Visual Studio .NET and the .NET Framework show the results.

Two of the components, ASP.NET and ADO.NET have their own chapters, so we revisit them again. We're sure that with practice, and the help of the other .NET-related books in the Syngress library, you'll find programming for the .NET Framework as fun as we have.

Solutions Fast Track

Obtaining the .NET Framework SDK

☑ Ensure that your computer meets the minimum requirements for running the .NET Framework. This will save you much time and trouble.

☑ www.microsoft.com/net is the main site for downloading.

☑ Follow the instructions carefully if you download the multiple small files. If you choose to download the one large file, give yourself plenty of time. It's about 120MB in size.

Installing the .NET Framework

☑ Remember that even though the .NET Framework is very stable and solid, any Beta software should be installed on noncritical machines.

☑ You can save a lot of time by making sure that your machine meets the minimum system requirements.

☑ The Visual Studio .NET install includes the .NET Framework SDK.

Common Language Runtime

- ☑ The CLR is the core runtime of the .NET Framework.
- ☑ Code that you write in any .NET language is compiled to the Microsoft Intermediate Language (MSIL).
- ☑ When first requested to be executed, the MSIL is then just-in-time–compiled to the applicable machine code for the CPU.

Developing Applications with the .NET Framework

- ☑ It is possible to write code with the .NET SDK using a simple text editor, and compile the code using the command line.
- ☑ ILDasm.exe allows you to see all of the types in your .NET executable. Double-clicking on any of the types will bring up a window displaying the MSIL code for that type.
- ☑ Because all .NET languages compile to MSIL, you can base your choice of language for .NET development on personal preference without taking a major performance hit.
- ☑ Using NGEN.exe on an assembly when you place it into production eliminates just-in-time compilation, which can improve start up performance.
- ☑ Become one with the Base Class Libraries. It's a lot of functionality for free.
- ☑ Take a hard look at structured exception handling.

Components in the .NET Framework

- ☑ The major components in the .NET platform utilize the services provided by the Base Class Library, and in most cases abstract that functionality to a higher level to make your life easier. An example would be the ASP.NET framework allowing you to program for the Web at a much higher level then would be necessary if you just used the Base Class Library.
- ☑ Windows Forms and ASP.NET Web Forms are the user interface frameworks for .NET development.

☑ ADO.NET provides the services for data access in the .NET Framework. It is optimized for performance (disconnected datasets) and productivity (object-oriented data access classes).

☑ Web services are components residing on a Web server accessible from anywhere using standard Web protocols.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: I installed Windows 2000 (Professional or Server), then installed the .NET Framework SDK. Why can't I create a Web project?

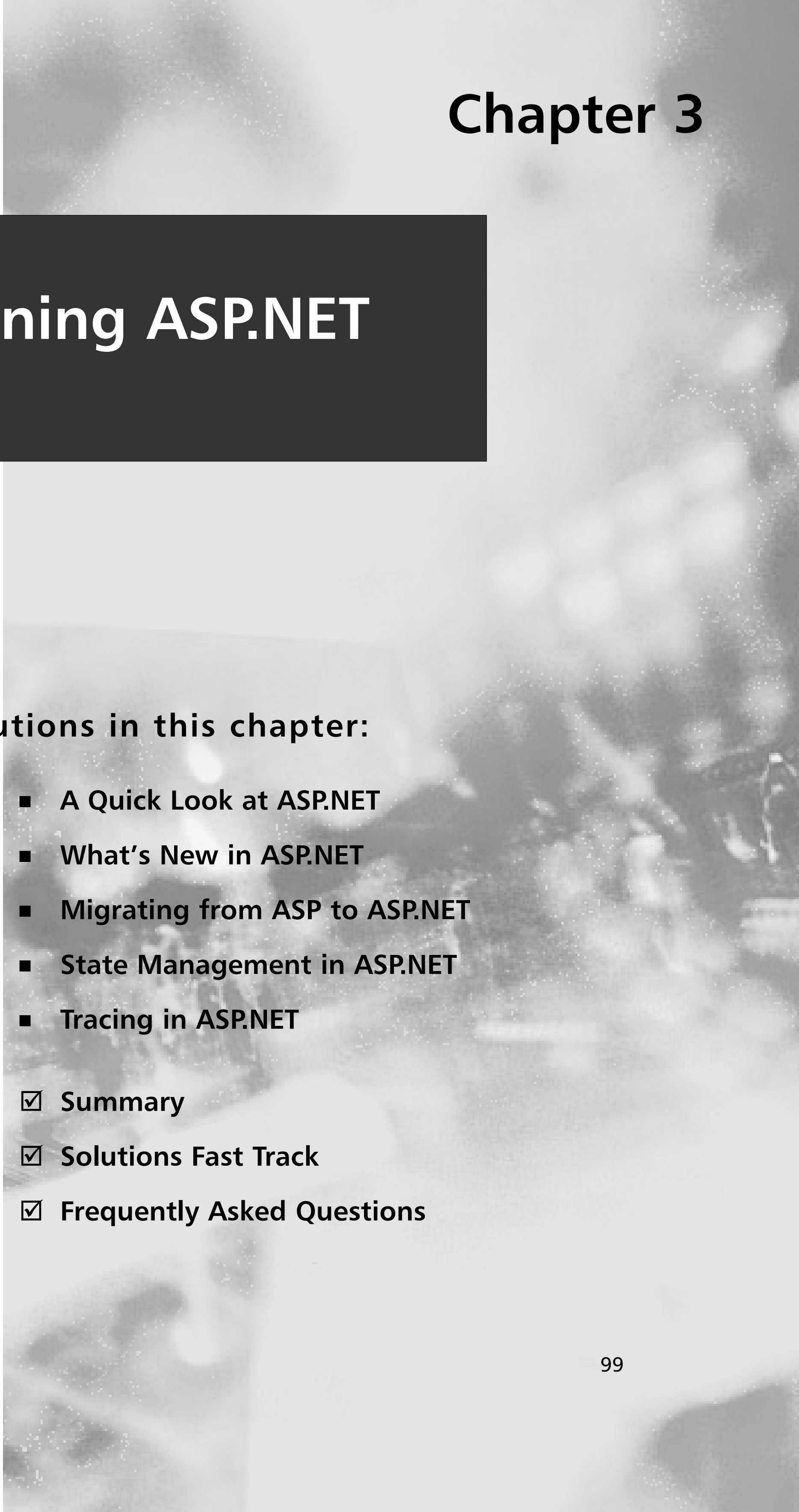
A: IIS 5.0 is a Windows component that isn't part of a normal Windows 2000 install unless you specifically set that as an install option. You must install IIS 5.0 prior to the .NET Framework SDK install, or ASP.NET won't register correctly. To recover, you must use regsvr32.exe to register the Aspnet_isapi.dll.

Q: I want to install the .NET Framework SDK on Windows NT 4.0 server, can I do that?

A: To install on Windows NT 4.0 Server, you must have service pack 6a applied.

Q: What is ASP.NET premium?

A: ASP.NET premium is a separate download and install giving you everything you need to develop ASP.NET applications. It includes the .NET Framework Redistributable, core ASP.NET support, and adds additional support for maintaining ASP.NET session state in a Web farm. In addition, there is support for output caching and secure hosting. It basically gives you more functionality that the ASP.NET support in the .NET Framework SDK. Possibly, by the time the .NET Framework SDK is out of beta and in production, this functionality will be integrated with the SDK.



Chapter 3

Learning ASP.NET

Solutions in this chapter:

- A Quick Look at ASP.NET
- What's New in ASP.NET
- Migrating from ASP to ASP.NET
- State Management in ASP.NET
- Tracing in ASP.NET

☑ Summary

☑ Solutions Fast Track

☑ Frequently Asked Questions

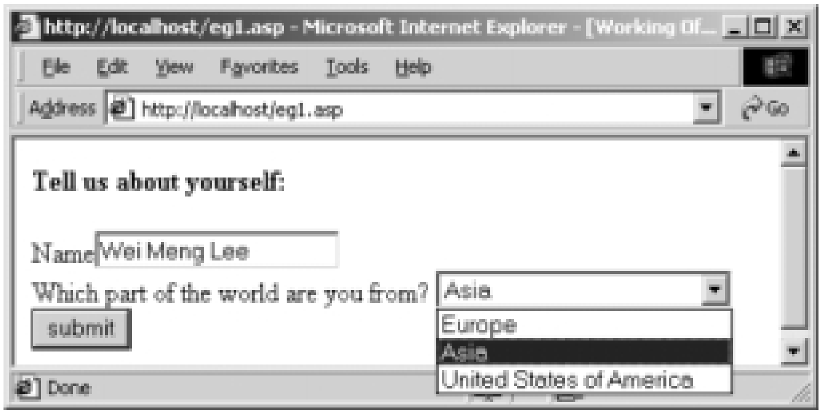
Introduction

A brand new version of ASP has hit the market and all you can think is "Great, now I need to learn a whole new development environment again." Well, true, although you have many new additions to learn in the new ASP.NET environment, not everything you learned in ASP development is gone. In fact, if you are familiar with ASP, concepts of the new ASP.NET programming environment will be very familiar to you. And, if you've never done any ASP programming, you can make use of the Visual Studio.NET environment during development, which will make learning very easy. This chapter gives you a high-level tour of ASP.NET and some of the tools available to you for development.

A Quick Look at ASP.NET

Before we look into the technical details of working with ASP.NET, let's consider a typical situation when writing a Web application with ASP. As you are probably aware, HTTP is a stateless protocol. Although being stateless has its benefits, such as reducing the resources on the server side, it often poses headaches for Web developers. The following example illustrates one of the problems with statelessness commonly faced by ASP developers. In this example, you have a form and you need users to fill it in and submit it for processing. The form may look like Figure 3.1.

Figure 3.1 Web Application Using ASP


http://localhost/eg1.asp - Microsoft Internet Explorer - [Working Of...
File Edit View Favorites Tools Help
Address http://localhost/eg1.asp
Go
submit
Done

Tell us about yourself:

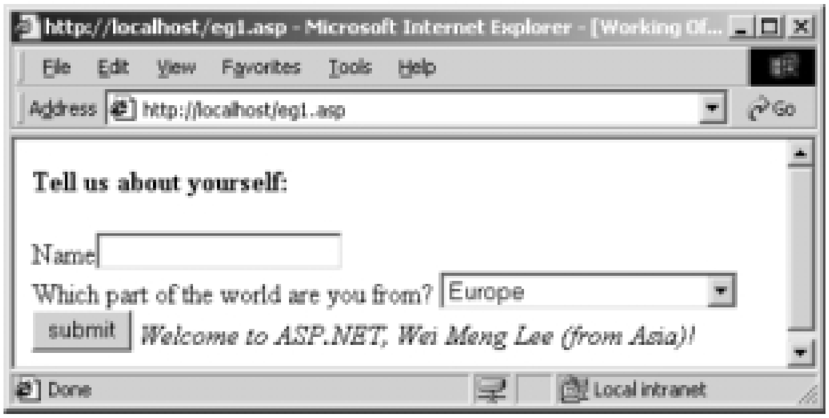
Name Wei Meng Lee

Which part of the world are you from? Asia

Europe
Asia
United States of America

When you click **Submit**, you print a message on the same page, as shown in Figure 3.2.

Figure 3.2 State Is Lost When the ASP Page Is Submitted


http://localhost/eg1.asp - Microsoft Internet Explorer - [Working Of...
File Edit View Favorites Tools Help
Address http://localhost/eg1.asp
Go
Done
Local intranet

Tell us about yourself:

Name

Which part of the world are you from? Europe

submit *Welcome to ASP.NET, Wei Meng Lee (from Asia)!*

Notice that the name in the text box is gone and the item in the selection list has been reset to the first item. Figure 3.3 shows the ASP code for the sample Web application.

Figure 3.3 Eg1.asp

```
<html>
<b>Tell us about yourself:</b>
<form method="post" action="eg1.asp">
Name<input type="text" name="userName"> <br/>
Which part of the world are you from?
   <select name="partOfWorld">
      <option value="Europe">Europe
      <option value="Asia">Asia
      <option value="United States of America">United States of America
   </select>
<input type="submit" value="submit">
<%
   if Request.Form("userName")<>""then
     Response.Write "<i>Welcome to ASP.NET,
         " & Request.Form("userName") & " (from
         " & Request.Form("partOfWorld") & ")!</i>"
   end if
```

Continued

Figure 3.3 Continued

```
%>
</html>
```

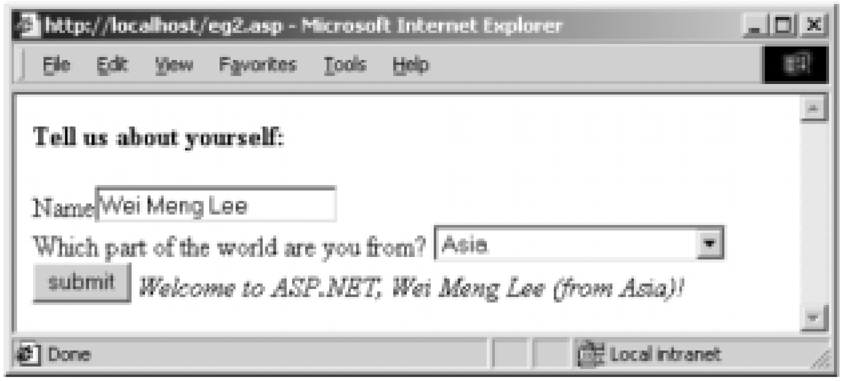
To ensure that the name and selection item remains selected after submission, you must modify the codes as shown in Figure 3.4.

Figure 3.4 Eg2.asp

```
<html>
<b>Tell us about yourself:</b>
<form method="post" action="eg2.asp">
Name<input type="text" name="userName" value="<% =
    Request.Form("userName") %>"> <br/>
Which part of the world are you from?
   <% itemSelected=Request.Form("partOfWorld") %>
   <select name="partOfWorld">
      <option <%if itemSelected="Europe" then Response.write
          "SELECTED" end if %>
value="Europe">Europe
      <option <%if itemSelected="Asia" then Response.write
          "SELECTED" end if %> value="Asia">Asia
      <option <%if itemSelected="United States of America"
          then Response.write "SELECTED" end if
%> value="United States of America">United States of America
   </select>
<input type="submit" value="submit">
<%
   if Request.Form("userName")<>""then
      Response.Write "<i>Welcome to ASP.NET, " &
      Request.Form("userName") & " (from " &
Request.Form("partOfWorld") & ")!</i>"
   end if
%>
</html>
```

And now the Web application will behave as you intend it to, as shown in Figure 3.5.

Figure 3.5 Preserving State in ASP Requires Substantial Effort


http://localhost/eg2.asp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Done
Local intranet

Tell us about yourself:

Name Wei Meng Lee

Which part of the world are you from? Asia

submit *Welcome to ASP.NET, Wei Meng Lee (from Asia)!*

From this simple example, you can see clearly the following problems with the current ASP technology:

- **Mixture of HTML and scripting codes** Our code in Figure 3.4 contains a mixture of display codes (HTML) and application logic (using VBScript). Because building Web applications often involves graphic designers *and* programmers, the current ASP technology does not provide a clean separation of display from content. This often results in bugs and difficulties in post-project maintenance.
- **Extra effort must be spent on maintaining states** In Figure 3.5, look at the amount of code you have to write in order for the server to maintain the state when transitioning from page to page. Most of the time spent on maintaining state could instead be directed toward implementing business logic.

NOTE

ASP.NET pages end with a .aspx extension.

Now let's look at how you can do the same thing using ASP.NET—look at the code shown in Figure 3.6.

Figure 3.6 Eg1.aspx

```
<script language="vb" runat="server">
   Sub Button1_clicked (sender as Object, e as EventArgs)
      message.text = "<i>Welcome to ASP.NET, " & userName.Text & "
          (from " & partOfWorld.Value & ") !</i>"
   End Sub
</script>
<html>
<body>
<form runat="server">
   <b>Tell us about yourself : </b><br/>
   Name : <asp:textbox runat="server" id="userName"/><br/>
   Which part of the world are you from?
   <select id="partOfWorld" runat="server">
      <option value="Europe"/>
      <option value="Asia"/>
      <option value="United States of America"/>
   </select>
   <asp:button runat="server" id="button1" onClick="Button1_clicked"
       text="Submit"/>
</form>
<asp:label runat="server" id="message"/>
</body>
</html>
```

The ASP.NET code shown in Figure 3.6 deserves closer attention. We can divide this code into two main parts, content and code. In the figure, the part related to code rather than content is depicted in boldface.

The Content Components

Within the user interface (UI) part of Figure 3.6, you can see familiar HTML code. In addition, you also see a few new tags starting with an *asp:* prefix. You might also notice that some of the elements have the additional *runat* attribute. Let's define some terms used in ASP.NET. The whole ASP.NET document shown in the example is known as a *Web Form*. A Web Form contains two components:

code and content. The content component of a Web Form can contain Web Form Server controls. Web Form Server controls contain the following types of controls: HTML Server control, ASP.NET Server control, Validation controls, and User controls. The examples in the next section illustrate the first two kinds of controls. (We examine the validation controls later in the chapter. User controls are much more complex and won't be addressed here.)

HTML Server Controls

An example of an HTML Server control is as follows:

```
<select id="partOfWorld" runat="server">
```

Notice that HTML Server controls are similar to the normal HTML elements, except that they have the additional *runat* attribute. In ASP.NET, normal HTML elements are converted to HTML Server controls so that they can be programmed on the server. The *id* attribute acts as a unique identifier for the server controls.

NOTE

If you have experience programming Visual Basic, a good way to view ASP.NET programming is to imagine yourself writing VB codes, except that this time your application runs on the Web platform. You can imagine an ASP.NET page as an executable file, producing HTML codes to be sent to the Web browser.

ASP.NET Server Controls

Besides the HTML server controls, ASP.NET provides a different set of server controls known as ASP.NET Server controls. You can think of ASP.NET Server controls as ActiveX controls in VB. Unlike the HTML Server controls, they do not provide a one-to-one mapping. The following is an example of an ASP.NET Server control:

```
<asp:button runat="server" id="button1" onClick="Button1_clicked"
    text="Submit"/>
```

This ASP.NET Server control will render itself as an *<input>* element when viewed using a Web browser. ASP.NET server controls expose properties and events that you can set and service. For example, this ASP.NET Server control defines the *onClick* event. When the button is clicked, the *Button1_clicked* subroutine would be serviced (which is covered in the next section).

NOTE

If you are experienced with HTML, think of ASP.NET server controls as another set of tags and elements that you can use to create dynamic Web applications. For example, instead of using the <input> tag for text input, you can also use the *<asp:input>* element (but with more features!).

The Code Components

The content component basically concerns itself with display issues. The code components, on the other hand, are the "glue" that binds things up. Your example shows a subroutine defined in the code section. This subroutine is fired when the user clicks the Submit button. It then displays a welcome message by referencing the controls defined in the content section:

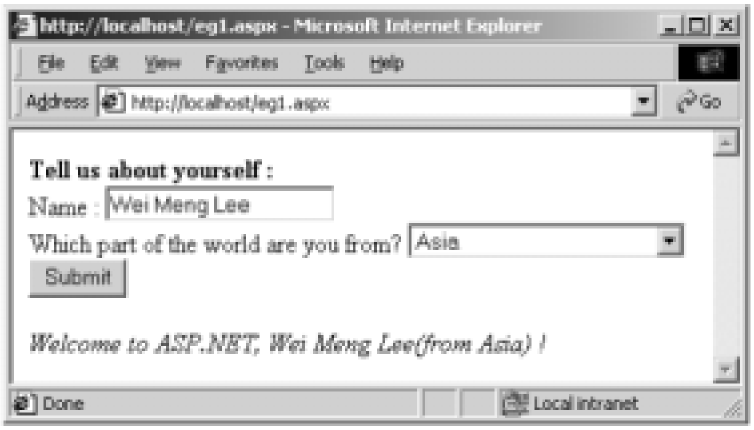
```
<script language="vb" runat="server">
   Sub Button1_clicked (sender as Object, e as EventArgs)
      message.text = "<i>Welcome to ASP.NET, " & userName.Text & "
          (from " & partOfWorld.Value & ") !</i>"
   End Sub
</script>
```

NOTE

The processing model of ASP.NET should be very familiar to VB programmers.

Figure 3.7 shows what the ASP.NET page looks like on the Web browser.

Figure 3.7 ASP.NET Preserves the State Automatically


http://localhost/eg1.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/eg1.aspx
Go
Done
Local intranet

Tell us about yourself :
Name : Wei Meng Lee
Which part of the world are you from? Asia
Submit

Welcome to ASP.NET, Wei Meng Lee(from Asia) !

After a page has been submitted, it will retain its state before the submission. To see the HTML codes generated by the ASP.NET runtime, select **View Source** (see Figure 3.8).

Figure 3.8 Your Example's HTML Output

```
<html>
<body>

<form name="ctrl1" method="post" action="eg1.aspx" id="ctrl1">
<input type="hidden" name="__VIEWSTATE"
value="YTB6OTY0MzM4NTkzX2Ewel9oejV6M3hfYTB6YTB6aHpUZVx4dF88aT5XZWxjb21lI
HRvIEFTUC5ORVQsIFdlaSBNZW5nIEx
lZShmcm9tIEFzaWEpICE8L2k+eF9feF9feHhfeF9feA==f77c15df" />

  <b>Tell us about yourself : </b><br/>
  Name : <input name="userName" type="text" value="Wei Meng Lee"
    id="userName" /><br/>
  Which part of the world are you from?

  <select name="partOfWorld" id="partOfWorld">
   <option value="Europe">Europe</option>
   <option selected value="Asia">Asia</option>
```

Continued

Figure 3.8 Continued

```
   <option value="United States of America">United States of
        America</option>
</select>

   <input type="submit" name="button1" value="Submit" id="button1" />

</form>
<span id="message"><i>Welcome to ASP.NET, Wei Meng Lee(from Asia)
    !</i></span>

</body>
</html>
```

What is interesting in this HTML output is the hidden input element, indicated in Figure3.8 in boldface. The *__VIEWSTATE* hidden element is the one that performs all the magic. It is responsible for "maintaining" states between pages. The value of this hidden element is used by the ASP.NET runtime to recall the previous state the page was in.

NOTE

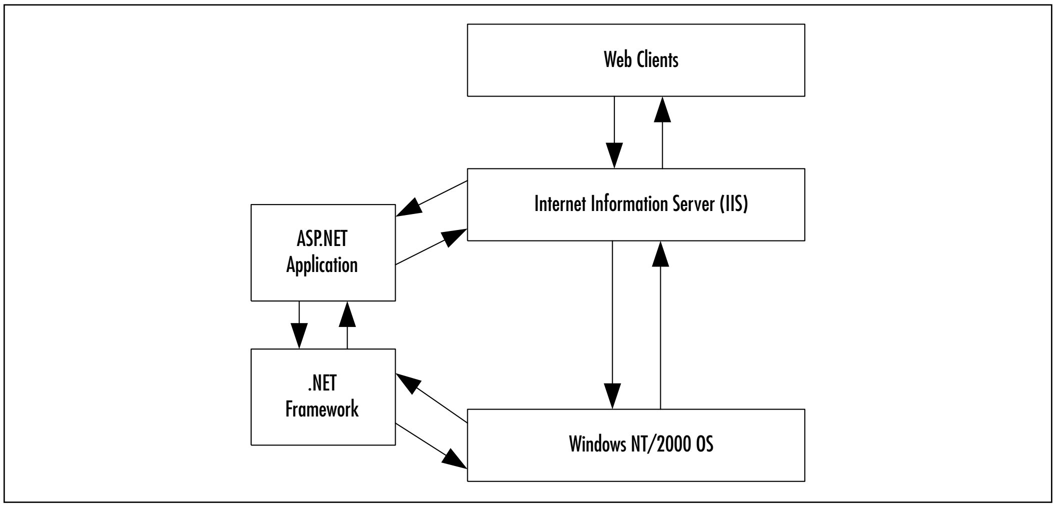
The concept of using a hidden element to maintain state is somewhat similar to that of using a browser session, with a *sessionid* passed as a hidden form value, or of using a cookie.

ASP.NET Architecture

Figure 3.9 illustrates the architecture of ASP.NET. The Web client first interacts with Internet Information Server (IIS). If the Web client is accessing HTML pages, IIS will communicate with the underlying operating system to fetch the HTML pages. If the Web client is accessing an ASP.NET application, the ASP.NET application will first be compiled to produce a .NET assembly.

One important difference between ASP.NET and ASP is that ASP.NET applications are parsed and compiled once and then cached, so that subsequent requests do not go through the same time-consuming steps. This creates a positive impact on the performance of ASP.NET applications.

Figure 3.9 Architecture of ASP.NET



What's New in ASP.NET

ASP.NET has revolutionized the way Web pages are built. No longer do you have to work with an interpreted language in which you have to write fairly structured code, but in an environment in which you can develop in response to users actions. ASP.NET has been rewritten from the ground up to make use of server-side controls, caching, better tracing mechanisms, and much, much more, all on top of an extensible and customizable framework.

The ASP.NET Environment

After examining Chapter 2, you should have a good idea of how the new .NET Framework and its supporting architecture works. ASP.NET is not just a programming environment but also an extension of the .NET Framework. ASP.NET is a collection of .NET classes specifically created to assist in the processing of HTTP requests and HTTP responses. As it sits on top of the Common Runtime Language (CLR), ASP.NET has access to the customization and extensibility features made possible in the .NET Framework.

While creating the ASP.NET programming model, Microsoft's goal was to revolutionize the way Web applications were written, and at the same time be able to utilize some of the concepts that current ASP developers are accustomed to. For instance, the ASP.NET environment still allows code and HTML to be intermixed on a page, but also has a new feature called code-behind, which provides a class module to separate the two.

ASP.NET has also been extended to work in conjunction with the Visual Studio.NET development environment. Note that Visual Studio.NET is not required to write ASP.NET applications. You as the developer are not tied to any specific development environment. However, the Visual Studio.NET environment provides an interactive development environment in which you can easily drag and drop controls onto forms, use property windows, use interactive debugging, view code or HTML, and much more. It is absolutely your choice to use a plain text editor, which doesn't assist in any way other than to keep you typing or to try the new ASP.NET design and development environment along with Visual Studio.NET. If you decide not to try the new IDE, you will need (at the very minimum) your editor of choice and ASP.NET redistributable.

Why Was ASP Rewritten?

ASP was rewritten from the ground up, not just improved upon, because after six years since its inception, it was time for an overhaul to make it state-of-the-art to meet demand for today's robust Web applications. ASP was based on earlier technology such as COM and is primarily a scripting language, which is hard to maintain. It needed to be easier to code, deploy, scale and have better development options and support for different browsers and devices.

Traditional ASP is based on an interpreted language, which by nature is slower than a compiled one. It also doesn't support or make use of strong types. Features such as caching and state management are available in ASP, but in a limited capacity and not well supported. ASP.NET provides a strongly typed class environment and strong support for caching and state management. A nice new feature of ASP.NET is the separation of code and HTML. If you have ever written an ASP application, you know of the nightmare code that results after coding and implementation of the user interface. ASP.NET provides a feature called *code-behind* to separate code and HTML for easier maintenance and enhancements. ASP.NET also provides interactive controls and tools for development, which to duplicate in ASP would require a mountain of code. For example, ASP.NET has server-side controls that you can drop onto a Web form and use instantly—no scripting involved. The amount of time saved by not having to script a text box and the corresponding validation code is significant.

Developing & Deploying…

Server-Side Controls

Server-side controls provide a means to develop Web applications similar to how traditional desktop applications using an event-driven programming model have been done in the past. The developer is no longer required to extract values from query strings and form data, as was done in ASP, but can directly reference the controls properties to retrieve values.

Another major improvement in ASP.NET and the .NET Framework is the support for browser compatibility. ASP.NET, as you will see in more detail in Chapter 4, can now detect the calling device and respond accordingly. This means that support for all types of browsing devices such as an Internet browser or cell phone or PDA is built into the framework. Taking that one step further, the Mobile Internet Toolkit along with ASP.NET will allow you to write one application to automatically render the user interface based on the calling device. The .NET Framework takes care of rendering the user interface for you whether it uses HTML or WML. But before you start writing mobile Internet applications, you need to have a decent understanding of how to write ASP.NET applications.

ASP.NET Language Support

The first thing to know about ASP.NET is the number of languages supported for development. No longer is VBScript and Jscript the only choice to build Web applications. ASP.NET supports the languages of .NET, most notably VB.NET, C#, and Jscript, as well as some third-party languages such as Perl, Python, and C++. Although, C++ is not directly supported for development in ASP.NET pages, you can use it for components that are referenced within a Web application. All .NET languages use and support a common set of base classes, are fully compiled, are object-oriented, and support inheritance including cross-language inheritance. With these features now available in any .NET language, you can now select the language you are most comfortable to work with.

NOTE

For all former ASP developers, VBScript will not be supported in ASP.NET; instead, it was replaced by full support of VB.NET.

Developing & Deploying…

Early-Bound Code

In languages such as Visual Basic.NET and Jscript.NET, writing late-bound code is very easy. Try to make use of the type-specific early-bound features that have been integrated into the languages. This is important because late-bound code will cause more execution and overhead for the Common Language Runtime during execution of a page.

Figure 3.10 shows early-bound code, which makes use of the namespace *System.Data.SqlClient*. The developer is able to directly reference the *SqlDataReader* as a result of the strongly typed object.

Figure 3.10 Early-Bound Code

```
  Imports System.Data.SqlClient

  Private Function GetEmployees() As SqlDataReader
Dim oSQLConn As New SqlConnection("Password=;User ID=sa; Initial
     Catalog=NetOverview;Data Source=localhost")
       Dim oSQLCmd As New SqlCommand()
       Dim oDR As SqlDataReader

       With oSQLCmd
            .CommandType = CommandType.Text
            .CommandText = "Select * from Employees"
            .Connection = oSQLConn
       End With
```

Continued

Figure 3.10 Continued

```
        oSQLConn.Open()
        oDR = oSQLCmd.ExecuteReader(CommandBehavior.CloseConnection)

        Return oDR
    End Function
```

Server Controls

One of the biggest new features included in the new ASP.NET Framework is server-side processing. You can now declare and program against controls on the server. The controls can even be event-driven from the client. This is a brand new concept from traditional ASP where the elements on a page are dependent upon their placement within script code, or the need for a developer to explicitly access the query string and regenerate HTML to retrieve values.

The most exciting news about server-side controls is that you can implement server-side processing with the addition of one line of code to a standard HTML control. The addition of the attribute *runat="server"*. The following shows an example of a simple input field made to process on the server:

```
<input id="Address1" type="text" runat="server">
```

Figures 3.11 and 3.12 show an example of simple controls that will transfer text from a text box to a label when the button is clicked.

Once you add the attribute, you can program against the control with server-side code. To provide even greater flexibility and more robust features, Microsoft created a new set of Server controls specifically for server processing, some of which are direct counterparts of HTML controls and other new ones we haven't had before to provide rich features for the UI.

Server controls allow you to adopt a programming model based around server-side event handling that is much more like the structured event-driven approach found in traditional executable programs. If you have ever developed in Visual Basic, this will be very familiar to you. If you haven't, think of it as coding the response to a user's action on a page. For example, if a user clicks a button on your page, you code a routine to handle the click event for that button. The routine is essentially the event of the control.

Figure 3.11 Simple Controls on a Web Page

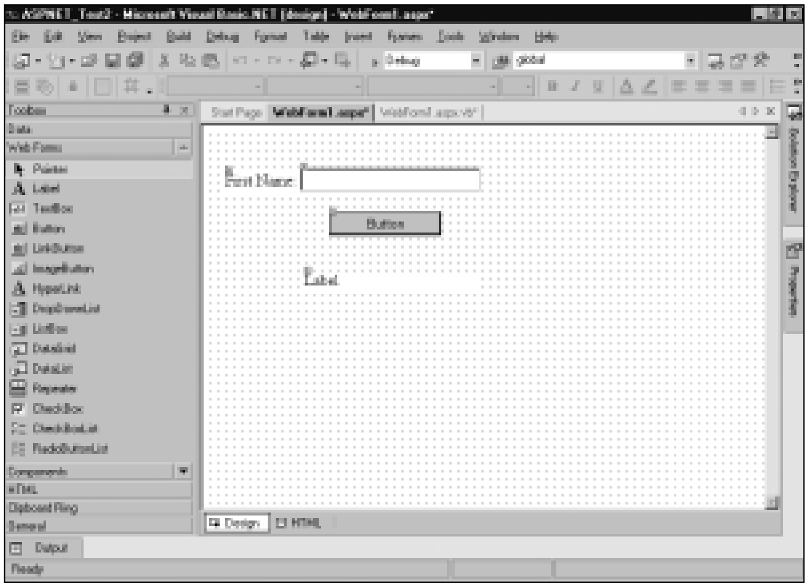
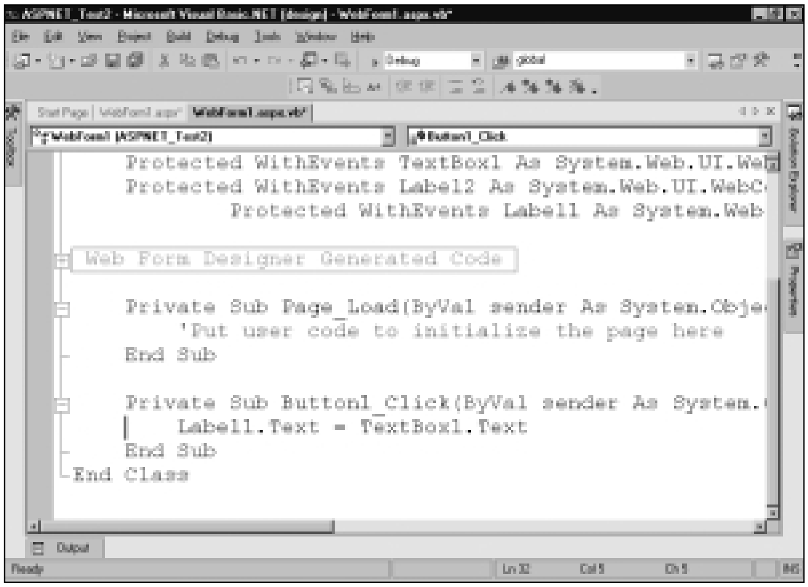


Figure 3.12 Code-Behind–Supported Simple Controls


ASPNET_Test2 - Microsoft Visual Basic.NET [design] - WebForm1.aspx.vb*
WebForm1 (ASPNET_Test2)
Button1_Click
Output
Ready
Ln 32
Col 5
Ch 5
INS

```
    Protected WithEvents TextBox1 As System.Web.UI.We
    Protected WithEvents Label2 As System.Web.UI.WebC
            Protected WithEvents Label1 As System.Web

 Web Form Designer Generated Code

    Private Sub Page_Load(ByVal sender As System.Obje
        'Put user code to initialize the page here
    End Sub

    Private Sub Button1_Click(ByVal sender As System.
        Label1.Text = TextBox1.Text
    End Sub
End Class
```

Another important point is that all controls are now considered objects in .NET. In ASP.NET, the whole page, including the controls and any HTML, is compiled into a class. Any controls or objects marked with the *runat="server"* attribute are created as objects within the page class. The objects' properties, methods, and events are now accessible by other objects through the page object. Items such as *Request*, *Response*, *Server*, *Application*, and *Session* are available directly through the *Page* class, instead of intrinsic values available to scripts in ASP. For example, the *IsPostBack* property of the page tells you when it's the first time a page has been requested or if it's a post-back from the client.

In traditional ASP, you would have referred to the form contents or copied the contents into variables in order to use them. In ASP.NET, you can directly reference the control by its ID and the property you are looking for. For example, if you want to get the text of my *FirstNameTextbox* and place the value into the text of a label, it would look something like this:

```
Label1.Text = FirstNameTextbox.Text
```

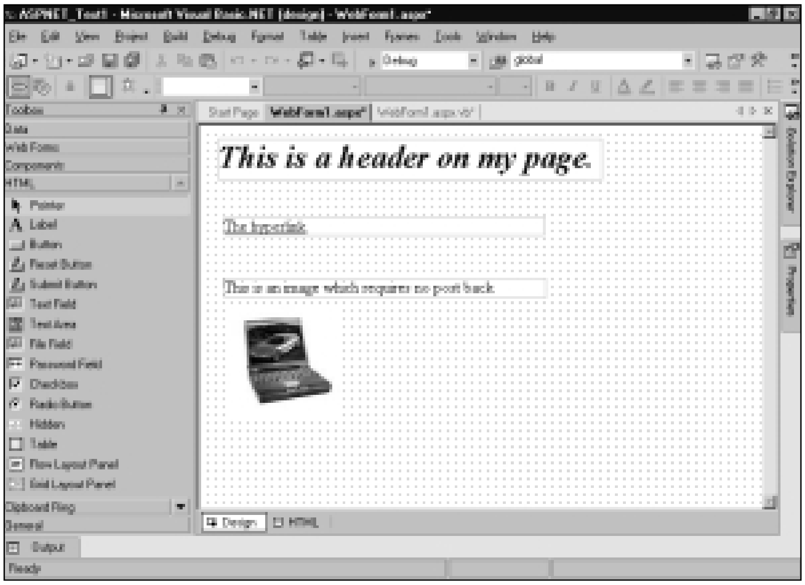
When to Use Server Controls

Up until now, it sounds as if a bunch of new controls provide server-side processing, and that you can freely start adding them to a Web page to create a Web application. The truth is that not every control should become a Server control. Remember, every time you invoke an event and write script to fire on the server, it will cause a post-back and round trip. The *runat="server"* attribute causes more overhead, making the right choice when using it on a Web page is important. In most cases, you do not want to create a server-side control in the following instances:

- The element is used only to run some client-side script.
- The element is a hyperlink that opens a different page.
- The element is simply static content that doesn't change.

In these cases, make use of the new ASP.NET HTML controls, which typically do not have the *runat="server"* attribute. Figures 3.13 and 3.14 show a sample of some HTML controls on a Web Form. Notice in Figure 3.14 how the supporting HTML is generated for the controls, specifically the lack of a *runat="server"* attribute.

Figure 3.13 HTML Controls



This is a header on my page.

The hyperlink

This is an image which requires no post back

Figure 3.14 The Supporting HTML to Display the HTML Controls

```
<%@ Page Language="vb" AutoEventWireup="false"
    Codebehind="WebForm1.aspx.vb"
    Inherits="ASPNET_Test1.WebForm1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
    <HEAD>
     <title></title>
     <meta name="GENERATOR" content="Microsoft Visual Studio.NET 7.0">
     <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
     <meta name="vs_defaultClientScript" content="JavaScript">
     <meta name="vs_targetSchema" content="http://schemas.microsoft.com/
     intellisense/ie5">
   </HEAD>
   <body MS_POSITIONING="GridLayout">
     <form id="Form1" method="post" runat="server">
      <DIV style="DISPLAY: inline; Z-INDEX: 101; LEFT: 19px;
          WIDTH: 383px;
```

Continued

Figure 3.14 Continued

```
          POSITION: absolute;
          TOP: 13px; HEIGHT: 42px" ms_positioning="FlowLayout">
         <FONT color="#000099" size="6"><EM><STRONG>This is a
          header on my page.</STRONG></EM></FONT>
       </DIV>
        <IMG style="Z-INDEX: 102; LEFT: 31px; POSITION: absolute;
          TOP: 179px" alt=""
          src="file:///E:\Documents and Settings\Administrator\Local
          Settings\Temporary Internet Files\
          Content.IE5\K12NO5MZ\149207[1].jpg">
       <DIV style="DISPLAY: inline; Z-INDEX: 103; LEFT: 25px;
          WIDTH: 321px; POSITION: absolute;
          TOP: 152px; HEIGHT: 21px" ms_positioning="FlowLayout">
          This is an image which requires no post back
       </DIV>
        <a href="www.microsoft.com">
        <DIV style="DISPLAY: inline; Z-INDEX: 104; LEFT: 25px;
         WIDTH: 320px; POSITION:
         absolute; TOP: 91px; HEIGHT: 19px" ms_positioning="FlowLayout">
         The hyperlink
         </DIV>
        </a>
    </form>
   </body>
</HTML>
```

Retaining State with Server-Side Controls

To retain state in ASP, the developer had to explicitly access the HTML Form field in the *Request.Forms* collection and then regenerate the HTML. The new ASP.NET server-side controls do this automatically. They make use of an *ID* field and a hidden _*VIEWSTATE* parameter on the specific control (see Figure 3.15). The _*VIEWSTATE* is used to store the state of a control that does not have its value posted back to the server. You do not need to program this, all of this is automatically created by .NET when the control is placed on a page. This will persist all the values in the controls across page loads. In Figure 3.16, you see what the *_VIEWSTATE* looks like for a label and text box on a Web page. The values that were entered in the text box are automatically encrypted in the *_VIEWSTATE*. Notice how the *_VIEWSTATE* was updated after the entry and execution of the text box and button (see Figures 3.17 and 3.18).

Figure 3.15 Simple Controls Displayed on a Browser

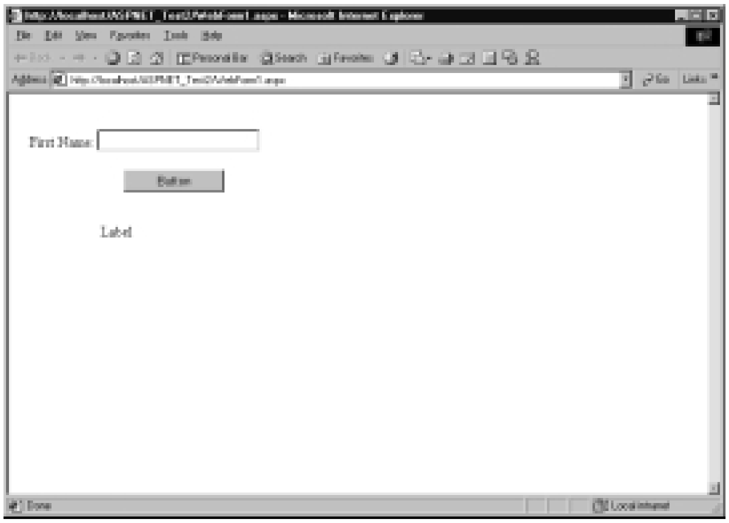
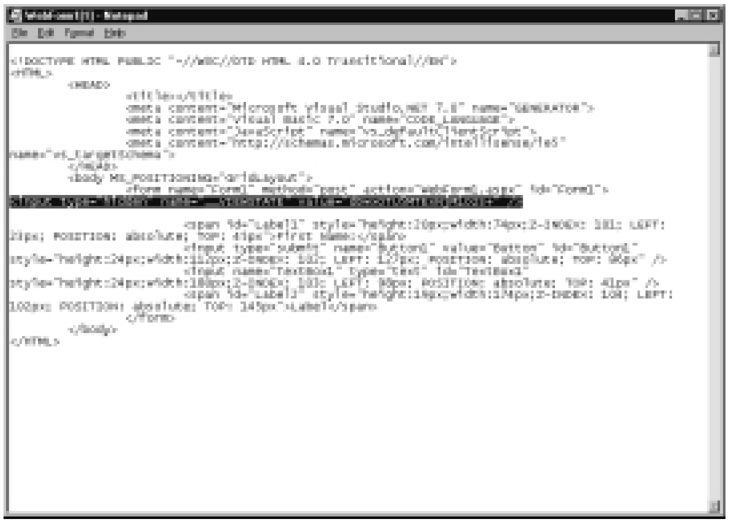


Figure 3.16 The *_VIEWSTATE* Generated by the Page



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
	<HEAD>
		<title></title>
		<meta content="Microsoft Visual Studio.NET 7.0" name="GENERATOR">
		<meta content="Visual Basic 7.0" name="CODE_LANGUAGE">
		<meta content="JavaScript" name="vs_defaultClientScript">
		<meta content="http://schemas.microsoft.com/intellisense/ie5"
name="vs_targetSchema">
	</HEAD>
	<body MS_POSITIONING="GridLayout">
		<form name="Form1" method="post" action="WebForm1.aspx" id="Form1">
<input type="hidden" name="__VIEWSTATE" value="[illegible]" />

			<span id="Label1" style="height:20px;width:74px;Z-INDEX: 101; LEFT:
23px; POSITION: absolute; TOP: 41px">First Name:</span>
			<input type="submit" name="Button1" value="Button" id="Button1"
style="height:24px;width:112px;Z-INDEX: 102; LEFT: 127px; POSITION: absolute; TOP: 96px" />
			<input name="TextBox1" type="text" id="TextBox1"
style="height:24px;width:188px;Z-INDEX: 103; LEFT: 98px; POSITION: absolute; TOP: 41px" />
			<span id="Label2" style="height:18px;width:174px;Z-INDEX: 104; LEFT:
102px; POSITION: absolute; TOP: 145px">Label</span>
		</form>
	</body>
</HTML>
```

Figure 3.17 Simple Controls Displayed after Entering a Value and Clicking on the Button

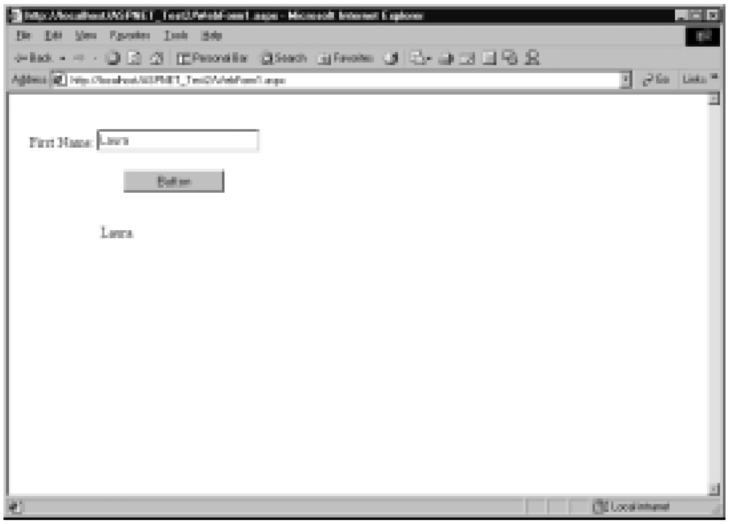
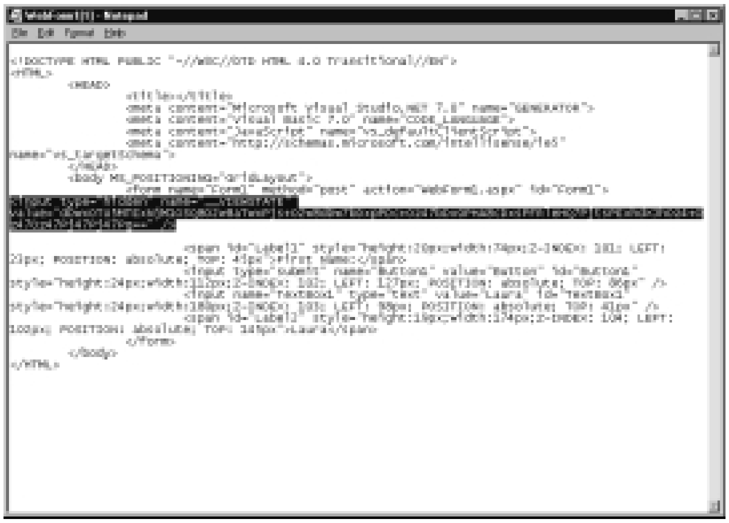


Figure 3.18 The _*VIEWSTATE* after Execution of the Page



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
        <HEAD>
                <title></title>
                <meta content="Microsoft Visual Studio.NET 7.0" name="GENERATOR">
                <meta content="Visual Basic 7.0" name="CODE_LANGUAGE">
                <meta content="JavaScript" name="vs_defaultClientScript">
                <meta content="http://schemas.microsoft.com/intellisense/ie5"
name="vs_targetSchema">
        </HEAD>
        <body MS_POSITIONING="GridLayout">
                <form name="Form1" method="post" action="WebForm1.aspx" id="Form1">
<input type="hidden" name="__VIEWSTATE"
value="[illegible]
[illegible]" />

                        <span id="Label1" style="height:20px;width:74px;Z-INDEX: 101; LEFT:
23px; POSITION: absolute; TOP: 41px">First Name:</span>
                        <input type="submit" name="Button1" value="Button" id="Button1"
style="height:24px;width:112px;Z-INDEX: 102; LEFT: 127px; POSITION: absolute; TOP: 86px" />
                        <input name="TextBox1" type="text" value="Laura" id="TextBox1"
style="height:24px;width:180px;Z-INDEX: 103; LEFT: 98px; POSITION: absolute; TOP: 41px" />
                        <span id="Label2" style="height:18px;width:174px;Z-INDEX: 104; LEFT:
100px; POSITION: absolute; TOP: 145px">Laura</span>
                </form>
        </body>
</HTML>
```

NOTE

Even though the _*VIEWSTATE* is encrypted, you should not use this for sensitive data, such as Logon ID or password. It is a simple form of encryption that can be unencrypted with a semi-sophisticated program.

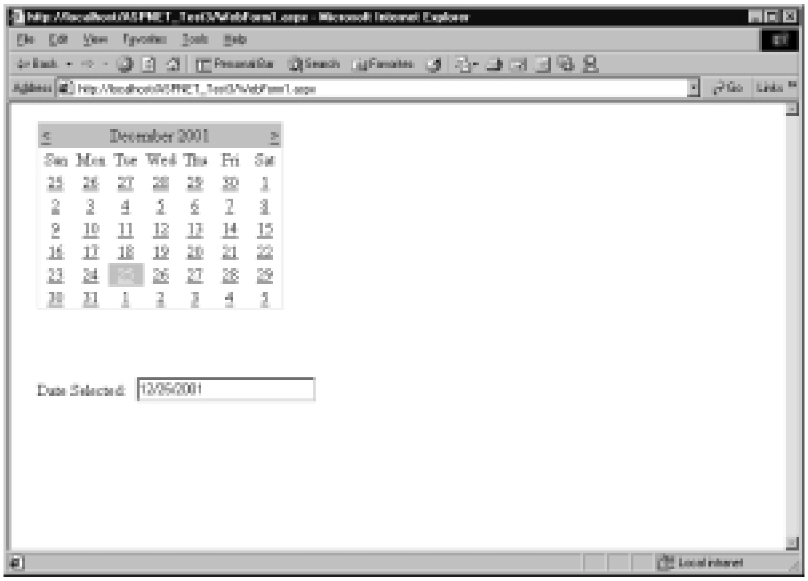
The encryption is used by the Web page to store data about the controls on your page. The encryption also makes it more difficult for code sniffers to determine values on a page, if they execute a View Source on a browser when in a Web page.

Web Controls

So what exactly are the new Web controls? ASP.NET created a set of primary controls that are similar to basic HTML Controls but are created to process server side and have been standardized to make them easier to code. Extra features have been added to these controls to reduce the amount of programming and make them a little more intelligent. For example, the *AutoPostBack* property is available on controls. If it set to true, this will prevent the control from being reset when the page is loaded. This is extremely important when you do not want data that has been entered or retrieved to change. The Web controls have been grouped based on the type of functionality they provide.

You can find the most common and often used set of controls in the Web Form controls group. They were created specifically for Web application development and provide a firmer object model and a higher degree of abstraction than traditional HTML controls. They include many of the HTML controls, as well as others with more functionality than is available with HTML elements. A good example is the *Calendar* control, which is in the Rich Control category. It renders a *Calendar* control that can retrieve a selected date with only one line of code. Figure 3.19 shows a *Calendar* control and the ability to retrieve a selected date. Notice how the *_VIEWSTATE* was updated after the entry and execution of the text box and button (see Figure 3.20).

Figure 3.19 Simple Controls on a Web Page


http://localhost/ASPNET_Test3/WebForm1.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost/ASPNET_Test3/WebForm1.aspx
Go Links
Local intranet

<			December 2001			>
Sun	Mon	Tue	Wed	Thu	Fri	Sat
25	26	27	28	29	30	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

Date Selected: 12/25/2001

Figure 3.20 The Code-Behind for the Calendar Control

```
Public Class WebForm1
    Inherits System.Web.UI.Page
    Protected WithEvents Calendar1 As
          System.Web.UI.WebControls.Calendar
    Protected WithEvents TextBox1 As
          System.Web.UI.WebControls.TextBox
    Protected WithEvents Label1 As
          System.Web.UI.WebControls.Label

    #Region " Web Form Designer Generated Code "

    'This call is required by the Web Form Designer.
    <System.Diagnostics.DebuggerStepThrough()>
    Private Sub InitializeComponent()

    End Sub
```

Continued

Figure 3.20 Continued

```
    Private Sub Page_Init(ByVal sender As System.Object, ByVal _
            e As System.EventArgs) Handles MyBase.Init
        'CODEGEN: This method call is required by the Web Form
         Designer
        'Do not modify it using the code editor.
        InitializeComponent()
    End Sub

#End Region

    Private Sub Page_Load(ByVal sender As System.Object, ByVal _
e As System.EventArgs) Handles MyBase.Load
        'Put user code to initialize the page here
    End Sub

    Private Sub Calendar1_SelectionChanged(ByVal sender As _
            System.Object, ByVal e As System.EventArgs) Handles _
            Calendar1.SelectionChanged
        TextBox1.Text = Calendar1.SelectedDate.ToShortDateString
    End Sub
End Class
```

One benefit behind the creation of Web controls is that they now share a similar group of attributes to make programming easier. This means that to change background color in a label or a text box or in a command button you use the *BackColor* property. In traditional ASP, the background color depended on the control being accessed. For example, you would modify the background color for a table element in HTML with the *BGColor* attribute, and for a span element, you would modify it with the *style* attribute. This meant that you had to know how to code the attributes for each type of control. The Web controls can be subdivided into categories of what they provide (see Table 3.1).

Table 3.1 Web Control Categories

Web Form Controls	List Controls	Rich Controls	Validator Controls
Label	*DropDownList*	*AdRotator*	*RequiredFieldValidator*
TextBox	*ListBox*	*Calendar*	*CompareValidator*
Button	*CheckBoxList*	*Xml*	*RangeValidator*
HyperLink	*RadioButtonList*		*RegularExpressionValidator*
LinkButton	*ListItem*		*CustomValidator*
Image	*Repeator*		*ValidationSummary*
Panel	*DataList*		
CheckBox	*DataGrid*		
RadioButton			
ImageButton			
Table			
TableRow			
TableCell			

- Web Form controls represent basic HTML elements.
- List controls represent HTML elements that produce grids or gridlike layouts.
- Rich controls will output rich content and complex functionality. An example is the *Calendar* control.
- Validation controls are new in ASP.NET and do not have an HTML representative in ASP. They perform validation on other controls.

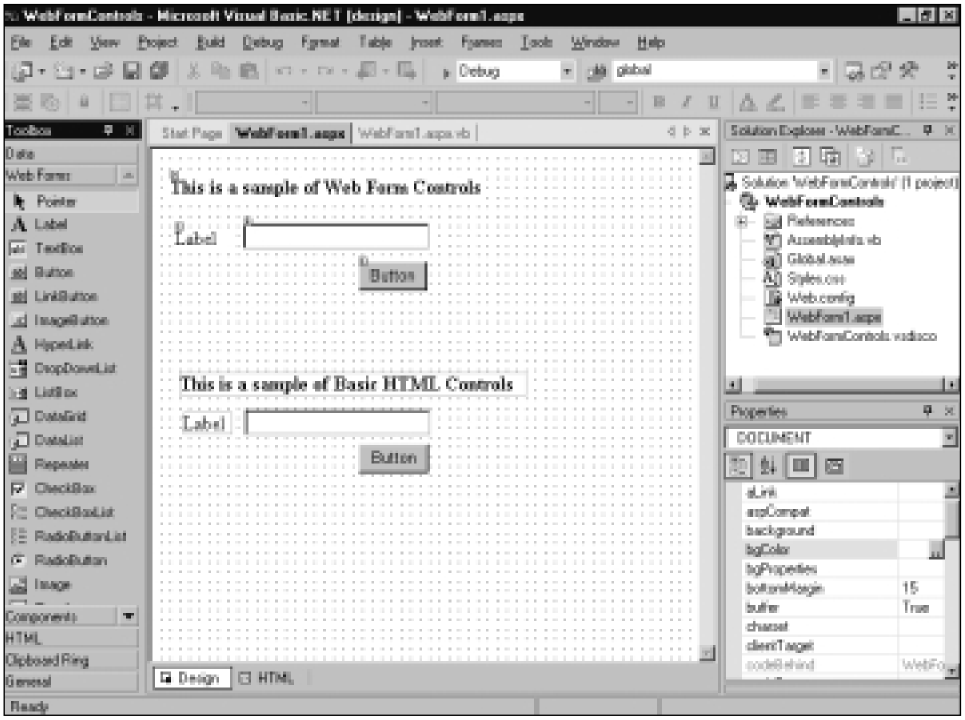
NOTE

All of the Web Form control types: *Basic*, *List*, *Rich*, and *Validator* are created with the *runat="server"* attribute when placed on a Web page. Only the HTML controls are created without the *runat="server"* attribute as the default and can be run with it set or not.

The last set of controls provided in the ASP.NET IDE are those which represent HTML elements, but in ASP.NET formats. They are grouped under HTML controls in the toolbox. Let's take a look at some Web controls and their ASP.NET HTML control counterparts to get an idea of some of the basic differences, even between the ASP.NET Web controls to the ASP.NET version of HTML controls. Keep in mind that the differences will be greater if we compare ASP.NET Web controls to straight HTML elements.

Let's take a look in design mode at some basic Web form controls. Notice in Figure 3.21 that the view they present is the same, even though the first set is created with *WebForm* controls and the second set with HTML controls.

Figure 3.21 Simple Controls on a Web Page



This is a sample of Web Form Controls

This is a sample of Basic HTML Controls

Now let's look at the HTML produced by both sets of controls (see Figure 3.22).

Figure 3.22 Representative HTML for the Controls Displayed in Figure 3.21

```
<%@ Page Language="vb" AutoEventWireup="false"
          Codebehind="WebForm1.aspx.vb"
```

Continued

Figure 3.22 Continued

```
           Inherits="WebFormControls.WebForm1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
     <HEAD>
          <title></title>
     </HEAD>
     <body MS_POSITIONING="GridLayout">
          <form id="Form1" method="post" runat="server">
               <asp:label id="Label1" style="Z-INDEX: 101;
                         LEFT: 16px; POSITION: absolute; TOP:
                         22px" runat="server" Width="263px"
                         Height="21px" Font-Bold="True">This is
                         a sample of Web Form Controls</asp:label>
               <asp:label id="Label2" style="Z-INDEX: 102;
                         LEFT: 20px; POSITION: absolute; TOP:
                         65px" runat="server" Width="45px"
                         Height="19px">Label</asp:label>
               <asp:textbox id="TextBox1" style="Z-INDEX:
                         103; LEFT: 77px; POSITION: absolute; TOP:
                         61px" runat="server"></asp:textbox>
               <asp:button id="Button1" style="Z-INDEX: 105;
                         LEFT: 173px; POSITION: absolute; TOP:
                         94px" runat="server"
                         Text="Button"></asp:button>

         <DIV style="DISPLAY: inline; FONT-WEIGHT:
                 bold; Z-INDEX: 108; LEFT: 24px; WIDTH:
                 289px; POSITION: absolute; TOP: 186px;
                 HEIGHT: 20px" ms_positioning="FlowLayout">
                    This is a sample of Basic HTML Controls
               </DIV>
               <DIV style="DISPLAY: inline; Z-INDEX: 109;
                         LEFT: 27px; WIDTH: 41px; POSITION:
```

Continued

Figure 3.22 Continued

```
                        absolute; TOP: 219px; HEIGHT: 19px"
                        ms_positioning="FlowLayout">
                    Label
               </DIV>
               <INPUT style="Z-INDEX: 110; LEFT: 78px;
                        POSITION: absolute; TOP: 217px"
                        type="text">
               <INPUT style="Z-INDEX: 111; LEFT: 173px;
                        POSITION: absolute; TOP: 246px"
                        type="button" value="Button">
          </form>
     </body>
</HTML>
```

The first item to notice between the two types of controls is the *ASP:* tag that is associated with every *WebForm* control. This is a new identifier tag in ASP.NET. Next, notice that the *WebForm* controls have all been identified with the attribute *runat="server"*. ASP.NET automatically (if using the Visual Studio.NET design environment) creates the control with this attribute. The ASP.NET HTML controls represent an idea of how you can convert traditional ASP scripted elements, but are also used for items that don't need to *POST* back to the server. Notice that they use <INPUT> tags and <DIV> tags you would see in traditional ASP. Also notice that both the ASP.NET *WebForm* controls and the ASP.NET HTML controls all reside within a <form> tag.

Let's look at Figure 3.23, which depicts the same page layout as it would in traditional ASP.

Figure 3.23 Sample ASP Code to Create a Form with a Text Message and Button

```
<%@ Language=VBScript %>

<html>
   <body>
      <p><b>This is a sample of classic ASP.</b> </p>
```

Continued

Figure 3.23 Continued

```
    <form id=form1 name=form1 method="post"
         action=OldASP.asp?mode=update>
  <p>Label <input id=txtMessage
                name=txtMessage></p>
       <input type=submit id=submit1 name=Button>
      </form>
   </body>
</html>
```

You can begin to see the levels of design translation from traditional scripted ASP, to ASP.NET HTML controls, to finally ASP.NET *WebForm* controls. All *WebControls* are derived from the *System.Web.UI.Control* class. ASP.NET adds the ASP: prefix to identify *WebForm* controls. When you get into the mobile aspects of creating mobile Web applications, you will notice that the Mobile Web controls use the *<Mobile:>* prefix instead of the *<ASP:>* tag. Refer to Chapter 4 for more information on the Mobile Web controls group and their use.

NOTE

The Visual Studio.NET environment provides you with drag-and-drop features for developing. When you drag a control onto a page, Visual Studio.NET will automatically set up default tags for the control, including a default ID and the *runat="server"* attribute. The Visual Studio design environment saves typing and will automatically provide a code-behind class for you to write content code in.

Separating Code—Inline or Code-Behind

In ASP, developers and designers often had to dance around each other to complete an application. That's because the code and HTML were intermingled on a page. The code was often messy to read and difficult to maintain. More often than not, a change from one team (developers) would crash something of the other team's (graphic design) and vice versa. The option to separate the two has been introduced in ASP.NET in two ways.

The first is code inline, as shown in Figure 3.24, where a developer can still put code in with the HTML but within a different section tag. The code will reside in a <script> tag and the HTML in a <body> tag. The following is a sample of a simple Web page created with inline code in ASP.NET. Note—this is the same way it was accomplished in ASP. The only difference is that in ASP.NET, you can have one <script runat="server"> tag to have code run on the server.

Figure 3.24 Inline Code

```
<html>
<!–This is a sample of an inline subroutine processed on the server–>

<script runat="server">

     Public Sub btn_Click(Sender as Object, E as EventArg)
     YourBirthDay.Text = BirthDay.Text
     End Sub

</script>



<!–This is a sample of the Inline content code processed on the server–>

<body>

     <form id="Form1" method="post" runat="server">

        Please enter your date of birth: <asp:TextBox
                  id="BirthDate" Runat="server"/>
     <br/>

     Click enter to save. <asp:Button OnClick="btn_Click"
          Runat="server" Text="Save"/>
```

Continued

Figure 3.24 Continued

```
        <br/>

        </form>

</body>
</html>
```

The second way to separate code is with code-behind. *Codebehind* is a separate class module that gets inherited into the page object. The *Codebehind* class makes use of page inheritance to separate the code logic from HTML presentation.

The actual *Codebehind* class is where the server-side code is stored. This allows different people to work on different aspects of the same page (content versus graphic design) without stepping on each other. To make use of the separate code file, the *@Page* directive will make use of the *Inherits* attribute for the class in which the code will reside.

So let's take the inline code from Figure 3.24 and create a *Codebehind* class for the code content. Figure 3.25 show an example of what the new HTML would look like using *WebForm* controls.

Figure 3.25 HTML for Web Form Controls

```
<%@ Page Language="vb" Codebehind="WebForm1.aspx.vb"
            Inherits="ProjectName.WebForm1"%>
<html>
<!—This is a sample of the Inline content code processed on the server—>

<body>

        <form id="Form1" method="post" runat="server">

            Please enter your date of birth: <asp:TextBox
                    id="BirthDate" Runat="server"/>
        <br/>
```

Continued

Figure 3.25 Continued

```
     Click enter to save. <asp:Button OnClick="btn_Click"
          Runat="server" Text="Save"/>
     <br/>

     </form>

</body>
</html>
```

Figure 3.26 is what the *Codebehind* class (WebForm1.aspx.vb) would look like.

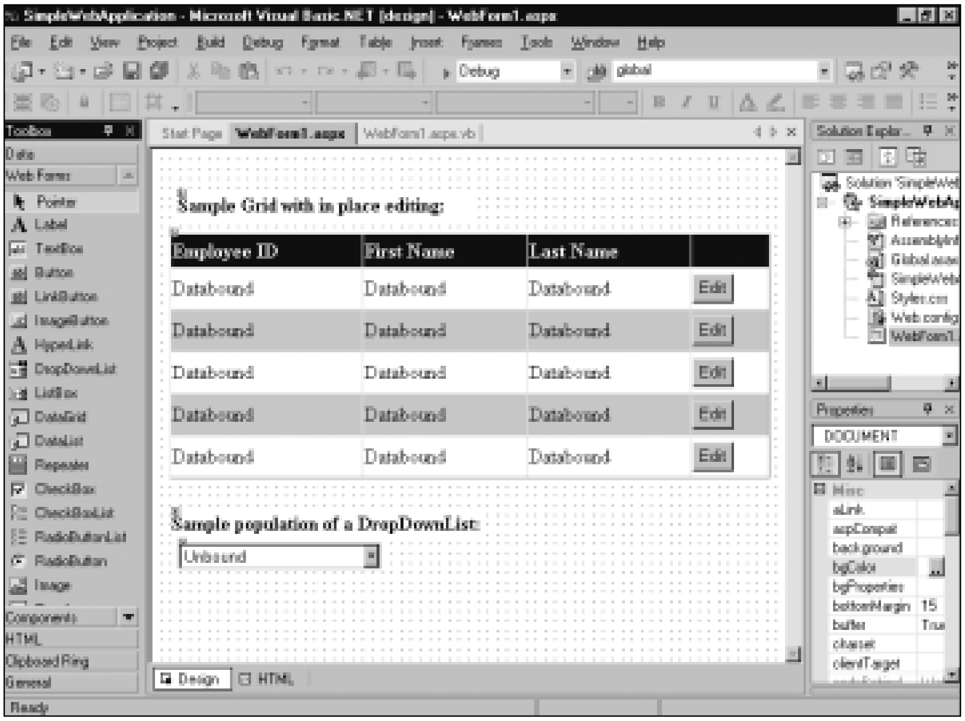
Figure 3.26 *Codebehind* Class

```
Imports System
Imports System.Web.UI
Imports System.Web.UI.WebControls

Public Class NameOfCodeBehindClass
      Inherits System.Web.UI.Page

     Public Sub btn_Click(Sender as Object, E as EventArg)
     YourBirthDay.Text = BirthDay.Text
     End Sub

End Class
```

So you can see that code-behind will take the presentation layer code out of the application code, which can make for much easier debugging and easier future enhancements.

We've created a grid control and a drop-down list and populated both with calls to a database to let you get a little more in-depth look at what's involved in populating some controls. Figure 3.27 shows the design-time controls.

Figure 3.27 Design-Time Drag and Drop of a *DataGrid* Control



Sample Grid with in place editing:

Employee ID	First Name	Last Name	
Databound	Databound	Databound	Edit
Databound	Databound	Databound	Edit
Databound	Databound	Databound	Edit
Databound	Databound	Databound	Edit
Databound	Databound	Databound	Edit

Sample population of a DropDownList:

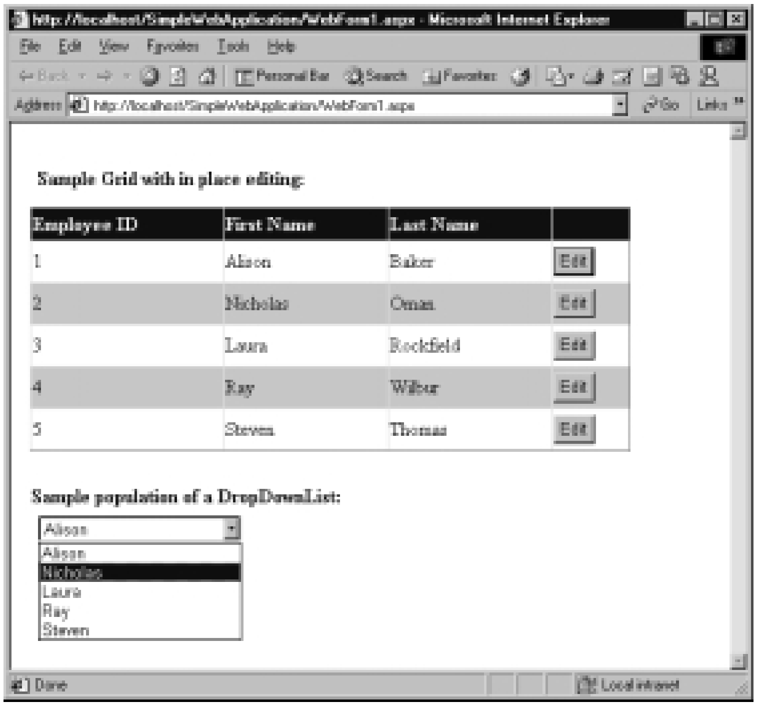
Unbound

For the grid, you can use some of the property builder fields to create the look of the alternating rows and the headers. You can also specify the columns you want to display and a command button that will perform inline editing. To use the property builder for this control, drag and drop the *DataGrid* control on a Web form, right-click on it, and select **Property Builder**. The Property Builder will allow you to add columns and set the headers. Use the *AlternatingItemStyle* property of the grid to set the background color for alternating rows.
When creating the button(s), add a command button of the type Edit, Update, Cancel from the Property Builder. Notice the HTML that is generated for the command button:

```
<asp:EditCommandColumn ButtonType="PushButton" UpdateText="Update"
    CancelText="Cancel" EditText="Edit"></asp:EditCommandColumn>
```

This sophisticated control takes care of modifying the view of the button when a user clicks on it. Your only responsibility is to code for the command events (Edit, Update, Cancel) on the grid. Refer to Figure 3.31 to see the processing code behind the command buttons. So, in Figure 3.28, let's see what it looks like when you run it.

Figure 3.28 Execution of *DataGrid*


http://localhost/SimpleWebApplication/WebForm1.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Personal Bar Search Favorites
Address http://localhost/SimpleWebApplication/WebForm1.aspx Go Links
Done Local intranet

Sample Grid with in place editing:

Employee ID	First Name	Last Name	
1	Alison	Baker	Edit
2	Nicholas	Oman	Edit
3	Laura	Rockfield	Edit
4	Ray	Wilbur	Edit
5	Steven	Thomas	Edit

Sample population of a DropDownList:

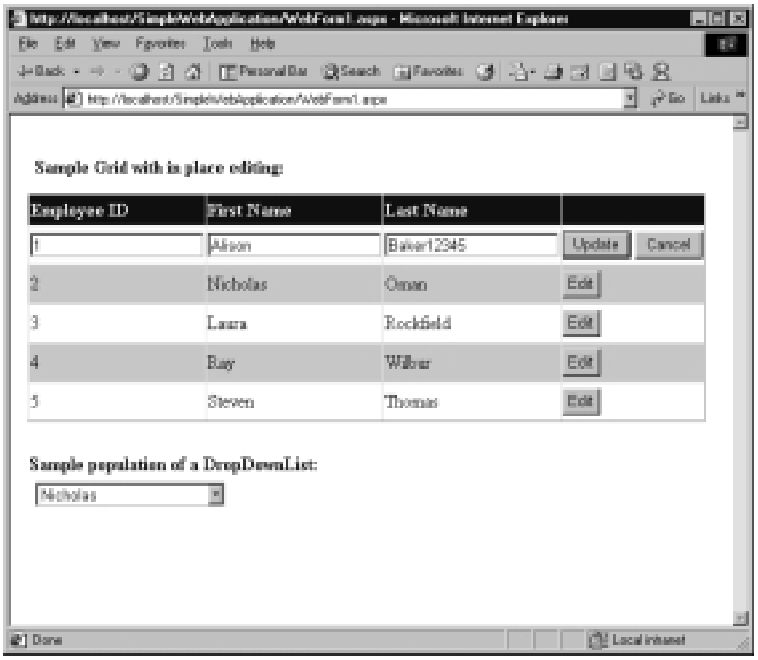
```
Alison
Alison
Nicholas
Laura
Ray
Steven
```

You can see that the rows in the grid have been populated and can be placed in edit mode when a user clicks the Edit button. The drop-down list has been populated with the first names available in the database. Figure 3.29 shows what happens when a user clicks the Edit button.

The row opens up for inline editing. If a user enters a new value and clicks Update, the grid shown in Figure 3.30 appears.

The row has been saved. Now let's take a look at the code that made it all happen. Essentially, two routines populate the *DataGrid* and the list box server controls. They are *GetEmployees()* and *FillNameDropDownList()*. They do the same thing but for two different tables. If you were to break this out into a more object-oriented architecture, you would normally put the database access into a separate class module (or programming tier), but that is a topic for another discussion.

Figure 3.29 Clicking a Button on the *DataGrid*



http://localhost/SimpleWebApplication/WebForm1.aspx - Microsoft Internet Explorer

Address http://localhost/SimpleWebApplication/WebForm1.aspx

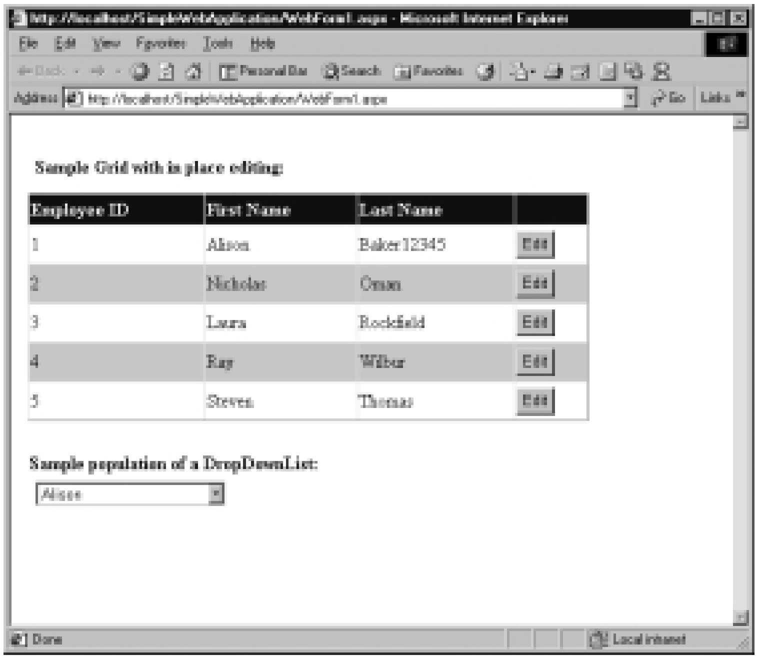
Sample Grid with in place editing:

Employee ID	First Name	Last Name	
1	Alison	Baker12345	Update Cancel
2	Nicholas	Oman	Edit
3	Laura	Rockfield	Edit
4	Ray	Wilbur	Edit
5	Steven	Thomas	Edit

Sample population of a DropDownList:

Nicholas

Figure 3.30 Save of *DataGrid* after Entering in Values



http://localhost/SimpleWebApplication/WebForm1.aspx - Microsoft Internet Explorer

Address http://localhost/SimpleWebApplication/WebForm1.aspx

Sample Grid with in place editing:

Employee ID	First Name	Last Name	
1	Alison	Baker12345	Edit
2	Nicholas	Oman	Edit
3	Laura	Rockfield	Edit
4	Ray	Wilbur	Edit
5	Steven	Thomas	Edit

Sample population of a DropDownList:

Alison

Three *DataGrid* events control the row Edit, Update, and Cancel functions for the grid. The routines are *dgEmployees_EditCommand()*, *dgEmployees_CancelCommand*, and *dgEmployees_UpdateCommand()*. The *Edit* command fills the grid. The *Cancel* command returns the grid to its initial state. The *Update* command saves an updated row to the database. All three use the *itemIndex* properties of the grid.

That's it! The code is very straightforward and easy to read. You don't have to plow through a bunch of HTML (as you would have in traditional HTML) to determine what has changed and how to handle it. The grid is event-driven and can determine which action to take based on the user's action upon it. To produce the same grid processing in ASP would have taken about three times as much code and about two days longer to create. Creating these two samples took around 30 minutes.

Figure 3.31 Code-Behind for Execution of the *DataGrid* for Figures 3.27 through 3.30

```
Imports System.Data.SqlClient


Public Class WebForm1
    Inherits System.Web.UI.Page
      Protected WithEvents Label1 As System.Web.UI.WebControls.Label
      Protected WithEvents Label2 As System.Web.UI.WebControls.Label
      Protected WithEvents ddlName As
          System.Web.UI.WebControls.DropDownList
      Protected WithEvents dgEmployees As
          System.Web.UI.WebControls.DataGrid


  #Region " Web Form Designer Generated Code "


   'This call is required by the Web Form Designer.
      <System.Diagnostics.DebuggerStepThrough()>
      Private Sub
         InitializeComponent()
      End Sub


   Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
```

Continued

Figure 3.31 Continued

```
       System.EventArgs) Handles MyBase.Init
      'CODEGEN: This method call is required by the Web Form Designer
      'Do not modify it using the code editor.
      InitializeComponent()
   End Sub

  #End Region

   Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
       System.EventArgs) Handles MyBase.Load

        'Retrieve the Employee data and set the grid if
        '  it is not a page postback
        If Not IsPostBack Then
            dgEmployees.DataSource = GetEmployees()
            dgEmployees.DataBind()
            FillNameDropDownList()
        End If
    End Sub

   Private Sub FillNameDropDownList()
      Dim oSQLConn As New SqlConnection("Password=;User ID=sa; Initial
         Catalog=NetOverview;Data Source=localhost")
       Dim oSQLCmd As New SqlCommand()
       Dim oDR As SqlDataReader
       Dim oNameItem As ListItem
       Dim bFirstTime As Boolean = True

       If ddlName.AutoPostBack = False Then
           'Fill with First Name
           With oSQLCmd
                .CommandType = CommandType.Text
                .CommandText = "Select fstName from Employees"
```

Continued

Figure 3.31 Continued

```
                .Connection = oSQLConn
          End With

          'Open the db connection and execute the select
          oSQLConn.Open()
          oDR = oSQLCmd.ExecuteReader(CommandBehavior.CloseConnection)

          'Fill the Drop down list
          ddlName.ClearSelection()
          Do While oDR.Read
              oNameItem = New ListItem(oDR.Item(0).ToString())
              If bFirstTime Then
                  oNameItem.Selected = True
                  bFirstTime = False
              End If
              ddlName.Items.Add(oNameItem)
          Loop

          'Close the DataReader
          oDR.Close()
      End If
  End Sub

  Private Function GetEmployees() As SqlDataReader
     Dim oSQLConn As New SqlConnection("Password=;User ID=sa; Initial
         Catalog=NetOverview;Data Source=localhost")
     Dim oSQLCmd As New SqlCommand()
     Dim oDR As SqlDataReader

     'Set the Command Text and Connection String
     With oSQLCmd
         .CommandType = CommandType.Text
         .CommandText = "Select * from Employees"
```

Continued

Figure 3.31 Continued

```
            .Connection = oSQLConn
        End With

        'Open and Fill the DataReader
        oSQLConn.Open()
        oDR = oSQLCmd.ExecuteReader(CommandBehavior.CloseConnection)

       'Pass the DataReader back to calling routine
         Return oDR
    End Function

    Private Sub dgEmployees_EditCommand(ByVal source As Object, ByVal e
          As System.Web.UI.WebControls.DataGridCommandEventArgs)
           Handles dgEmployees.EditCommand

         'Fill the employees
        dgEmployees.DataSource = GetEmployees()

         'Set the EditItemIndex to the row being edited
         dgEmployees.EditItemIndex = e.Item.ItemIndex
         dgEmployees.DataBind()
    End Sub

   Private Sub dgEmployees_CancelCommand(ByVal source As Object, ByVal
        e As System.Web.UI.WebControls.DataGridCommandEventArgs)
        Handles dgEmployees.CancelCommand

       'Fill the employees
       dgEmployees.DataSource = GetEmployees()

       'Reset the grid
       dgEmployees.EditItemIndex = -1
       dgEmployees.DataBind()
```

Continued

Figure 3.31 Continued

```
    End Sub

   Private Sub dgEmployees_UpdateCommand(ByVal source As Object, ByVal
      e As System.Web.UI.WebControls.DataGridCommandEventArgs) Handles
      dgEmployees.UpdateCommand

      Dim oFirstNameCell As TableCell = e.Item.Cells(1)
      Dim oLastNameCell As TableCell = e.Item.Cells(2)
      Dim oIDCell As TableCell = e.Item.Cells(0)
      Dim iEmpID As Integer =
         System.Convert.ToInt32(CType(oIDCell.Controls(0),
         TextBox).Text)
      Dim strFirstName As String = CType(oFirstNameCell.Controls(0),
         TextBox).Text
      Dim strLastName As String = CType(oLastNameCell.Controls(0),
           TextBox).Text

      'Call the update routine passing grid row information
      Call UpdateEmployee(iEmpID, strFirstName, strLastName)

      'Reset the grid
      dgEmployees.DataSource = GetEmployees()
      dgEmployees.EditItemIndex = -1
      dgEmployees.DataBind()
    End Sub

    Private Sub UpdateEmployee(ByVal EmpID As Int32, ByVal FirstName As
       String, ByVal LastName As String)

      Dim oSQLConn As New SqlConnection("Password=;User ID=sa; Initial
         Catalog=NetOverview;Data Source=localhost")
      Dim oSQLCmd As New SqlCommand()
```

Continued

Figure 3.31 Continued

```
      With oSQLCmd
          .CommandType = CommandType.Text
          .CommandText = "Update Employees set lstName = '" & LastName
              & "', fstName = '" & FirstName & "' where ID = " &
              EmpID.ToString
          .Connection = oSQLConn
      End With

       'Execute the query that does not return any rows
       oSQLConn.Open()
       oSQLCmd.ExecuteNonQuery()
       oSQLConn.Close()
   End Sub

End Class
```

So, you can now start to see the power of Web Form controls and event-driven programming in ASP.NET Web applications. It's straightforward, robust, and fairly flexible.

Configuration Files

The configuration files help govern how an application should act. The configuration files are XML-based, which make them more readable and easier to change. Two types of configuration files are used in the upcoming Web applications: the *Machine.Config* and the *Web.Config*.

The *Machine.Config* file contains the defaults for all ASP.NET applications on the machine in which ASP.NET has been installed. In traditional ASP, configuration was handled by script written to update the IIS metabase or through a series of screens and manual setting in the IIS Manager. The *Machine.Config* is a great place to store information that needs to be consistent across all ASP.NET applications running on that server. The *Machine.Config* file will play a large role in mobile Web application development, as you will see later in Chapter 4, because it houses device-specific information that you can use in any mobile application.

The *Web.Config* contains the information for a specific ASP.NET application. Settings applied in the *Web.Config* will override those set in the *Machine.Config*, which will allow for customization in individual Web applications. (*Note:* You can lock down settings in the *Machine.Config* so that the Web.config file cannot override them). The Web.config file is a great place to store database connection information or to write a custom handler for your application. A great example of a custom handler that you may use for mobile devices is one that identifies the calling device and directs the request to the appropriate page or routine to handle it.

Among its other features, the *Machine.Config* supports XCOPY, which is used for deployment. It is what gets you away from the dreaded DLL nightmares you have probably experienced in prior Web and client/server application development. One bonus of .NET is that components are self-registering. All that is required is that they are copied to their target location. The CLR is able to load multiple versions of the assembly at the same time. This is called *shadow copying* and essentially gets rid of "DLL hell." Of course, there is much more to deployment, but the scope of that discussion is too broad for this chapter. For now, know that registering of every component and application is no longer required.

Migrating from ASP to ASP.NET

You can't just simply migrate code from ASP to ASP.NET and have it work, unless it's the most basic of Web applications. Yes, by renaming the .asp extension to .aspx you have created an ASP.NET page, but it remains to be seen if it will actually do anything. Remember, ASP.NET has been completely rewritten to provide a superior development environment, which means that some of the things you did in ASP aren't supported in ASP.NET, and for good reason. This doesn't mean that everything you previously did won't work, just a good majority of it. Microsoft did a good job of trying to maintain some of the ASP syntax but could only do so in situations where they were not detrimental to application performance and success in ASP.NET.

Note that ASP can run side-by-side of ASP.NET applications. This is because ASP.NET does not use any of the libraries or processes used for ASP. ASP.NET has its own environment to run in, including the new page extension (.aspx), the request handler (ASP_WP.EXE), and the configuration file (Web.config).

The Global.asa file has also been updated to the Global.asax in ASP.NET. Just like its predecessor, it is still found in the root directory of the application. Both the Global.asa and Global.asax can reside in the same root directory if needed. This is so that existing ASP pages can run along side ASPX pages.

If you do decide to convert your ASP to ASP.NET pages, remember that you cannot share state between the two types of development. In most cases, you will have to convert all your ASP pages at once to get them to work together within a project. There is no way to convert just one page of an ASP application to ASP.NET and have it work with the remaining ASP pages.

We wish there was an easy way of converting the pages, but there isn't. If you are going to try, we suggest making copies of the originals and renaming them to the ASPX version. Pull them into a text editor or Visual Studio.NET and begin converting tags and processing as needed. We recommend starting out with the basic HTML controls provided by ASP.NET. They are not server-side processing controls, but you can convert them by adding the *runat="server"* attribute.

For a majority of your ASP applications, writing a new representative ASPX page from scratch is a better idea than trying to convert the existing ASP page. You may find that the new environment will provide a better architecture for your current ASP application and that redesigning it for ASP.NET will improve overall performance and scalability.

Figure 3.32 shows a quick example of an ASP page that will verify if a field has had a text entry. Notice that it includes many *Response.Write* methods and the use of JavaScript to validate that the field has had something entered.

Figure 3.32 Validation Routine in ASP

```
<%@ Language=VBScript %>

<%
     If Request.QueryString("mode") = "update" then
     'The user entered something and submitted the page
          dim sMessageText

          sMessageText = Request.Form("txtMessage")
          Response.Write ("<html><body>")
          Response.Write ("Hello ")
          Response.Write (sMessageText)
          Response.Write ("</body></html>")

     else
     'this is the users first time in
```

Continued

Figure 3.32 Continued

```
%>
          <html>
          <body>

<p><b>Welcome to ASP "Classic".</b>  <br>Please enter your name.</p>

<form id=form1 name=form1 method="post" action=OldASP.asp?mode=update
    onsubmit="return form_validate(this)">
              <input id=txtMessage name=txtMessage>
              <input type=submit id=submit1 name=submit1>
          </form>

          </body>
          </html>

<%
     end if
%>

<script language="JavaScript">

function form_validate(theForm)

     {
          if (theForm.txtMessage.value == '')
               {
                    alert('Please supply your name');
                    return false;
               }
         else
          {
                    return true;
               }
```

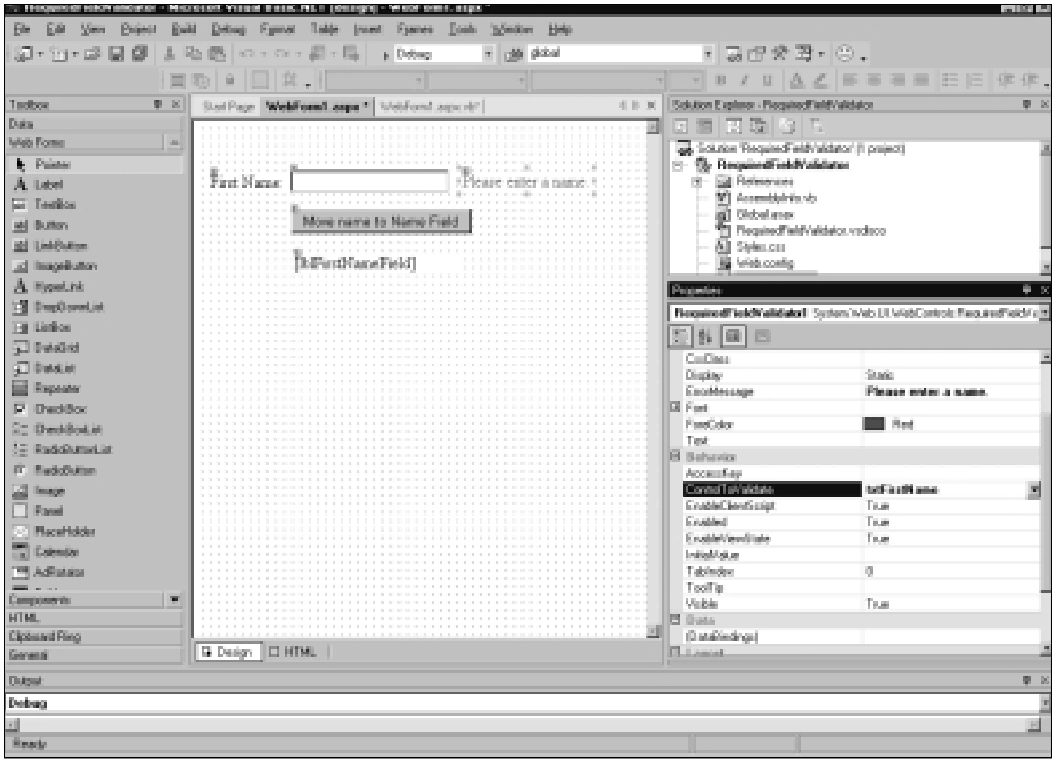
Continued

Figure 3.32 Continued

```
     }
</script>
```

Now, let's write the same functionality for ASP.NET using a *RequiredFieldValidator* control. See Figure 3.33.

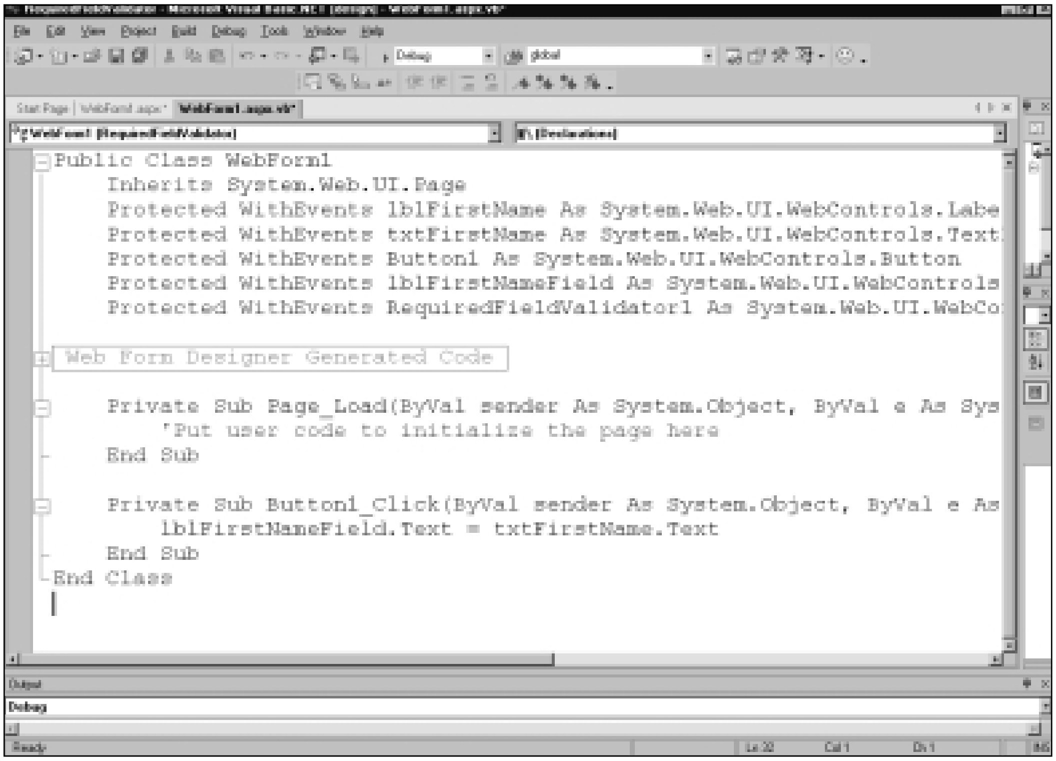
Figure 3.33 Required Field Validator



CssClass	
Display	Static
ErrorMessage	Please enter a name.
Font	
ForeColor	Red
Text	
Behavior	
AccessKey	
ControlToValidate	txtFirstName
EnableClientScript	True
Enabled	True
EnableViewState	True
InitialValue	
TabIndex	0
ToolTip	
Visible	True
Data	
(DataBindings)	

Notice that you can use ASP.NET with the Visual Studio.NET design environment. You can drag a label, text box, and a *RequiredFieldValidator* control on the page. Using the Properties window for the *RequiredFieldValidator* control, you can select which control to validate—in this case, the *txtFirstName* text box. You can set the *ErrorMessage* to the text you want displayed when the control is validated and returns that the field is invalid. That's it! You don't have to write any code or script. To prove it, we've included a screenshot of the code-behind (see Figure 3.34).

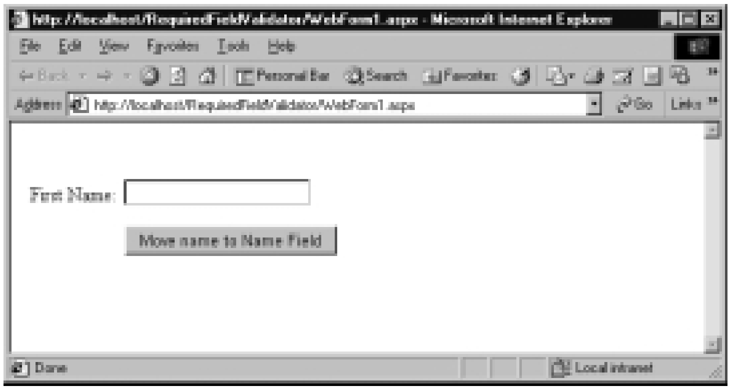
Figure 3.34 Validator Control Code-Behind



```
Public Class WebForm1
    Inherits System.Web.UI.Page
    Protected WithEvents lblFirstName As System.Web.UI.WebControls.Labe
    Protected WithEvents txtFirstName As System.Web.UI.WebControls.Text
    Protected WithEvents Button1 As System.Web.UI.WebControls.Button
    Protected WithEvents lblFirstNameField As System.Web.UI.WebControls
    Protected WithEvents RequiredFieldValidator1 As System.Web.UI.WebCo

 Web Form Designer Generated Code

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As Sys
        'Put user code to initialize the page here
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
        lblFirstNameField.Text = txtFirstName.Text
    End Sub
End Class
```

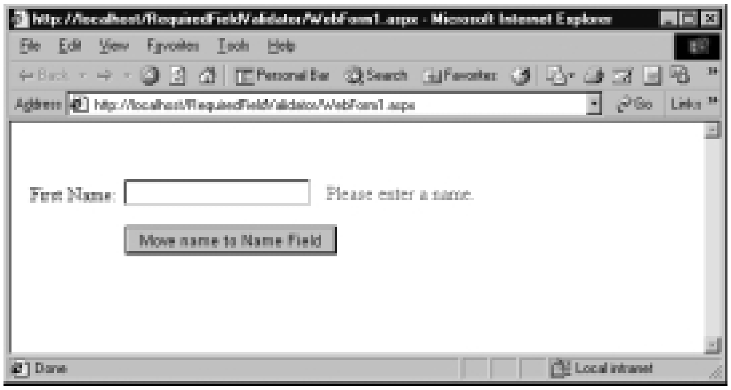
Notice the only code entered is the move from the text box first name to the label for first name. No scripting required. When we initially run the page we should see the output shown in Figure 3.35.

Figure 3.35 Simple Web Page with Validator Control that Has Not Been Fired


http://localhost/RequiredFieldValidator/WebForm1.aspx - Microsoft Internet Explorer
First Name:
Move name to Name Field
Done
Local intranet

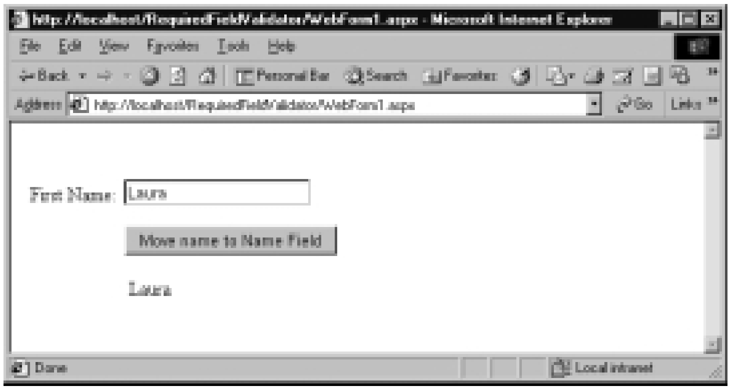
Clicking on the button without making an entry will produce the screen shown in Figure 3.36.

Figure 3.36 Simple Web Page with Validator Control that Has Been Fired Because No Text Was Entered


http://localhost/RequiredFieldValidator/WebForm1.aspx - Microsoft Internet Explorer
First Name:
Please enter a name.
Move name to Name Field
Done
Local intranet

Finally, entering a name and then clicking the button results in the output shown in Figure 3.37.

Figure 3.37 Simple Web Page after Valid Entry of Text


http://localhost/RequiredFieldValidator/WebForm1.aspx - Microsoft Internet Explorer
First Name:
Laura
Move name to Name Field
Laura
Done
Local intranet

As you can see, you saved an enormous amount of time by using the ASP.NET *RequiredFieldValidator* control instead of writing script code to perform the same function. This is why, in most cases, you wouldn't want to directly port over your ASP code. You will have to decide how to achieve the most bang for your buck when converting.

State Management in ASP.NET

One of the most challenging aspects of Web application development is maintaining state. You can think of state as the condition of the information used in the application at any given point in time. ASP.NET has done a great job of improving state management over what is available in traditional ASP development.

Application State

Application state is maintained through the *Application* object and is available to all resources accessible to a given ASP.NET application. You can use Application state to help reduce the cost associated with expensive data retrieval. An example would be to retrieve data once, store it in Application state, and then retrieve it from there when needed instead of calling the database. A simple example of using Application state is as follows:

```
' Create and set an Application level value
Application("ProjectName") = "MyProjectName"

' Use Application level value
Textbox1.text = Application("ProjectName")
```

However, although it can be beneficial in reducing overhead for retrieval of information, you have to be very careful when implementing it for data manipulation purposes. Any application that can be accessed by several users at the same time and is using it for data manipulation is at risk for concurrent access issues. You can access Application state through the HTTP *Application* class using the *Application* property. To reduce the risk of concurrency issues, use the *Lock()* method of the *HttpApplicationState* class. This will prevent anyone else from updating or using the Application state at the same time. Just remember when you are done to release the lock by calling the *Unlock()* method. Figure 3.38 shows a simple example of implementing the lock and unlock methods of the *Application* class.

Figure 3.38 Programming the Lock and Unlock Method for Application State

```
'Create and set an Application level value
Application("CounterField") = 0

'Use Application level value - lock and unlock
```

Continued

Figure 3.38 Continued

```
Application.Lock()
Application("CounterField") = Application("CounterField") + 1
Application.UnLock()
```

NOTE

Application state is not optimized as an "update" environment or for use in a Web farm or Web garden. Application state is ideally tied to a single instance of an application. Trying to share access to the application's state from different machines is not possible because it is tied to a single application.

Session State

Session state is used more often than Application state for the simple reason that in most Web applications, you like to store and use data for a particular user, not just the application. You use session state to store data for a given user during a Web session.

ASP.NET takes advantage of session variables by automatically creating a *sessionID* when a Web application is invoked. The Web server creates the *sessionID* upon the initialization of the Web application. This automatically created *sessionID* is passed back and forth between the user's client and the server during server post-backs, each time using the *sessionID* to bind information to that user. It is this *sessionID* that will allow multiple pages within a Web application to know information about a user without having to perform any query lookups.

To program for Session state, you create variables just like you saw in Application state, replacing the *Application* state for *Session*. For example:

```
' Create and set a Session level value
Session("ProjectName") = "MyProjectName"

' Use Session level value
Textbox1.text = Session("ProjectName")
```

This can be a powerful tool when passing information from page to page within an application. It reduces the need for making a trip to the database to retrieve information that was just provided in a previous page in the application. However, keep in mind when using session variables that you don't go overboard with them. They do use memory to function, and the use of too many at one time may cause performance problems within the application.

Wasn't State in ASP?

In traditional ASP, Session state is available only for the life of the application and not across machines. So a Web farm using traditional ASP session variables is not able to share information across servers. Also, traditional ASP made use of HTTP cookies to share session information between an application and client. This method worked great, but only if the client would accept cookies. Lastly, all traditional ASP requests are serialized, meaning that if the process request's mechanism is shut down, the Session state will be shut down as well, losing information.

Out-of-Process

ASP.NET took care of being able to share session state across servers by implementing out-of-process Session state. There are two types: StateServer and SQL Server.

StateServer

StateServer Session state is where data is stored in memory in a process separate from where ASP.NET is running. It can be on the same server or on a different server. To create a StateServer session, you need to first configure the machine that will store the session information. Start by making sure that the ASP.NET State service is started. You do this by first finding the ASP.NET State service on the machine. You can find it by going to **Start | Settings | Control Pane**l and opening up the **Administrative Tools** window. Once in that window, click on **Services**. You can also use the Microsoft Management Console (MMC), which you can find by going to **Start | Programs | Administrative Tools | Computer Management**. Once there, view the Services found under the Services and Applications choice. In either case, look for ASP.NET State, right-click it, and select **Start**. It includes many settings, but they are beyond the scope of this chapter.

Once ASP.NET State is turned on, you need to tell the application to use it. You can do this by updating the Web.config file of the application. Set the mode of <*SessionState*> = *"StateServer"* and the *stateConnectionString* to tell ASP.NET which server and port to use. A sample Web.config file is as follows:

```
<sessionState
        mode="StateServer"
        stateConnectionString="tcpip=127.0.0.1:42424"
 />
```

The default port for ASP_NET is 42424, as shown in the preceding code. Note that if you want multiple servers to be able to access the same session, you will need to make sure that the IP address of the server reflects the same IP address where the ASP.NET state service is running. Additionally, each server will have to have the same machine key in order to identify and use the session information. The machine key is used to create encrypted data and the *sessionID*. In order for each server to understand and use the different session information, they each need to use the same one.

NOTE

When preparing to use another machine to store session state be cognizant of possible performance hits. It's a great option to have a fully supported Session state server that can be used by multiple clients, but not if it becomes a detriment to application performance.

SQL Server

SQL Server Session is just like that of StateServer but uses a relational database as the storing mechanism. You need to perform configuration as you did for the StateServer session but in SQL language. To configure SQL Server to manage session, you need to run a Transact SQL (T-SQL) script on the server. The script comes with ASP.NET and is named InstallSQLState.sql. You can find the directions for installing it in Visual Studio.NET Help.

Again, you will need to update the Web.config file, this time with the SQL Connection string. Notice that the mode of the <SessionState> has been changed to *SQLServer* and a *sqlConnectionString* added:

```
<sessionState
      mode="SQLServer"
```

```
            sqlConnectionString="data source=127.0.0.1;
                user id=session;password=5aBc00"
      />
```

Selecting the Mode

So now you have read about Windows Service state and SQL Server state, but how do you know when to use them? You can choose among four types, and all are set through the Web.config file in the *sessionState* section. See Table 3.2 for the details of the four types.

Table 3.2 Mode Types

Mode Type	Description
Off	This means that there is no state management in the application.
InProc	This is the default for all new Web applications and means that session will be run in-process, which is faster than out-of-process. This should be used when Session data will not be shared among servers or when it is not critical to the success of an application.
StateServer	This is an out-of-process mode that is great when multiple servers are requesting Session information. Reliability is greater because a single process is managing the state for multiple servers.
SQLServer	This is best when data is the critical path for an application. This type of sessionState is usually slower than the others.

Setting Timeout

In all cases for state management, you can set the Session *timeout* parameter. This is the duration for which the session will be cleaned out and no longer available until reset. In the following example, the *timeout* is set for 100 minutes:

```
<sessionState
        mode="inproc"
        timeout="100"
/>
```

ASP.NET now provides many options and support for state management in applications. Choosing the right kind for the application you are building can be tricky, but if done correctly can increase application performance significantly.

Cookieless Session

ASP.NET improved upon ASP session state by creating a way to support cookieless sessions. This is accomplished through the process of URL *munging*. Instead of storing the *sessionID* in a cookie, it is munged in with the URL and passed back and forth. The receiving page can decode the munged URL and retrieve the session information from it. In order to implement cookieless mode for an application, you need to update the Web.config file. The normal default is *cookieless="false"*. Figure 3.39 shows a simple example of an entire Web.config file with Session state showing *cookieless="true"*

Figure 3.39 Sample Web.Config file with the Session State Section Highlighted

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>


  <system.Web>


  <!—  DYNAMIC DEBUG COMPILATION
   Set compilation debug="true" to insert debugging symbols
   (.pdb information) into the compiled page. Because this creates
   a larger file that executes more slowly, you should set this
   value to true only when debugging and to false at all other times.
   For more information, refer to the documentation about debugging
   ASP.NET files.
    —>
   <compilation defaultLanguage="vb" debug="true" />


   <!—  CUSTOM ERROR MESSAGES
     Set customErrors mode="On" or "RemoteOnly" to enable custom
        error messages, "Off" to disable.
     Add <error> tags for each of the errors you want to handle.
```

Continued

Figure 3.39 Continued

```
   —>
  <customErrors mode="RemoteOnly" />

   <!—  AUTHENTICATION
    This section sets the authentication policies of the application.
        Possible modes are "Windows",
        "Forms", "Passport" and "None"
   —>
    <authentication mode="Windows" />



    <!—  AUTHORIZATION
        This section sets the authorization policies of the application.
        You can allow or deny access
        to application resources by user or role. Wildcards: "*" mean
        everyone, "?" means anonymous
        (unauthenticated) users.
    —>
    <authorization>
        <allow users="*" /> <!— Allow all users —>

        <!—  <allow       users="[comma separated list of users]"
                          roles="[comma separated list of roles]"/>
               <deny      users="[comma separated list of users]"
                          roles="[comma separated list of roles]"/>
        —>
  </authorization>

  <!—  APPLICATION-LEVEL TRACE LOGGING
   Application-level tracing enables trace log output for every page
   within an application. Set trace enabled="true" to enable application
```

Continued

Figure 3.39 Continued

```
   trace logging.  If pageOutput="true", the trace information will be
   displayed at the bottom of each page.  Otherwise, you can view the
   application trace log by browsing the "trace.axd" page from your Web
   application root.
  —>
  <trace enabled="false" requestLimit="10" pageOutput="false"
   traceMode="SortByTime" localOnly="true" />



  <!—  SESSION STATE SETTINGS
   By default ASP.NET uses cookies to identify which requests belong to a
   particular session. If cookies are not available, a session can be
   tracked by adding a session identifier to the URL. To disable cookies,
   set sessionState cookieless="true".
  —>
  <sessionState
          mode="InProc"
          cookieless="true"
  />

  <!—  PREVENT SOURCE CODE DOWNLOAD
   This section sets the types of files that will not be downloaded.
   As well as entering a httphandler for a file type, you must also
   associate that file type with the xspisapi.dll in the App Mappings
   property of the Web site, or the file can be downloaded. It's recommended
   that you use this section to prevent your sources being downloaded.
  —>
  <httpHandlers>
   <add verb="*" path="*.vb" type=
       "System.Web.HttpNotFoundHandler,System.Web" />
   <add verb="*" path="*.cs"
type="System.Web.HttpNotFoundHandler,System.Web" />
```

Continued

Figure 3.39 Continued

```
   <add verb="*" path="*.vbproj" type=
      "System.Web.HttpNotFoundHandler,System.Web" />
   <add verb="*" path="*.csproj" type=
       "System.Web.HttpNotFoundHandler,System.Web" />
   <add verb="*" path="*.webinfo" type=
       "System.Web.HttpNotFoundHandler,System.Web" />
  </httpHandlers>


  <!—  GLOBALIZATION
   This section sets the globalization settings of the application.
  —>
  <globalization requestEncoding="utf-8" responseEncoding="utf-8" />

  </system.Web>
```

Choosing between Cookies and Munged URLs

Note that choosing between the use of cookies and munged URLs should be done at the application level. Cookies are more efficient than munged URLs but are not supported on every browser, and client users can arbitrarily turn them off. Vice versa, munged URLs can work on any browser but are less efficient than cookies. An application cannot use both, so you need to make the decision up-front. Also keep in mind while making the decision that munged URLs do not work with absolute URLs.

Caching

ASP.NET didn't stop at state management but created a facility to act as a holding area for data and information outside of the available Application and Session state. You now have a fully integrated and supported cache facility. The idea behind caching was to create a more robust Application-like state paradigm, but with added features, such as the ability to control item addition and removal.

Caching can improve efficiency in any ASP.NET application and is performed in three ways. The first two: page output and page fragment caching involve a page directly by caching the code that generates the page. The code is run only when necessary, therefore saving overhead. The last type helps to avoid round trips by utilizing the worker process on the back-end data source to cache to the page. Caching is ideal for items or objects that can be retrieved on an as-needed basis. Caching can be performed throughout the Web application or on the client by the browser, on a Web server by page or data, or in between them by proxy. The two types, browser and proxy, are not directly controlled by ASP.NET because they deliver content from the client machine or proxy server. Because of this, they will reduce Web server traffic. However, the last type, Web server by page or data caching, is directly supported by ASP.NET.

Page Caching

To improve the performance of dynamically generated pages, use page output caching. This type of caching works by dynamically generating the page the first time. During the first access, the page will be cached. Any future requests for the page will use the cached version to render it. This will save the time it would have taken to dynamically render it. This type of caching is best suited to pages that are primarily static.

To control the duration at which a page is automatically refreshed, you can set the *OutputCache* directive. This is set within the HTML at the page level. Notice the simple Web page that contains only a Label and a Grid. Notice in Figure 3.40 that the only line added was the *OutputCache* duration near the top.

Figure 3.40 *OutputCache*

```
<%@ Page Language="vb" AutoEventWireup="false"
     Codebehind="Caching.aspx.vb" Inherits=
          "Productivity.Caching"%>
<%@ OutputCache duration="10" VaryByParam="*" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<title></title>
</HEAD>
<body MS_POSITIONING="GridLayout">
  <form id="Form1" method="post" runat="server">
     <asp:DataGrid id="dgData" style="Z-INDEX: 101; LEFT: 70px;
```

Continued

Figure 3.40 Continued

```
                POSITION: absolute; TOP: 102px" runat="server"
                Width="438px" Height="178px"></asp:DataGrid>
          <asp:Label id="Label1" style="Z-INDEX: 102; LEFT: 72px;
                POSITION: absolute; TOP: 49px" runat="server"
                Width="272px" Height="20px">Label</asp:Label>
     </form>
  </body>
  </HTML>
```

By setting the *duration = 10*, you are setting the page to be refreshed every ten seconds.

NOTE

A great example to try is to place a label on a Web page and populate it with the system clock displaying the time with the seconds shown. Set the *OutputCache duration = "10"*, as shown in Figure 3.40. Now, run the Web page and continuously click the Refresh button on the browser. Notice that the time will be updated every 10 seconds only.

Notice in the preceding example that the *VaryByParam* property = "*". The *VaryByParam* property allows you to set the number of cached page versions. The asterisk specifies that every combination of the querystring or *POST* request be cached. Other settings include none or specific variables. Setting the value to "None" says that only one version of the *GET* request will be cached.

```
VaryByParam="none"
VaryByParam="*"
VaryByParam="LastName, EyeColor"
```

Table 3.3 defines the three other attributes that you can use to define caching durations.

Table 3.3 Other Attributes that You Can Use to Define Caching Durations

Attribute	Description
VaryByHeader	Caching is performed when the header string changes from one request to another.
VaryByControl	Caching is performed for specific controls that you identify. This is great for page section caching. Headers and menu bars are a prime candidates for page section caching. This is done by creating a user control and setting the OutputCache duration and the VaryByParam. Every page after the first page will use the cached version of the control.
VaryByCustom	Allows you to define your own set of criteria for caching.

Data Caching

Another way to improve performance is to limit the number of round trips to the data server. Similar to page caching, data can be retrieved and cached for future use. Instead of calling the database every time data is needed, you can have the cache supply it. Again, this works great for data that is used often and needs to be refreshed but doesn't change frequently. Just think how much overhead is saved by not making the database connection and round trip to the data server.

Figure 3.41 depicts a simple example of checking the cache first for data before making the call to the data server. If the database is called, you can save the information to the cache at the same time.

Figure 3.41 Checking the Cache for Data

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
            System.EventArgs) Handles MyBase.Load

         Dim oDS As DataSet
         oDS = Cache.Get("Employess")

         If IsNothing(oDS) Then
              oDS = New DataSet()
              SqlConnection1.Open()
              SqlDataAdapter1.Fill(oDS, "Employees")
              SqlConnection1.Close()
```

Continued

Figure 3.41 Continued

```
            Cache.Add("Employees", oDS, Nothing, DateTime.MaxValue, New
                TimeSpan(0, 0, 30), Caching.CacheItemPriority.Default,
                Caching.CacheItemPriorityDecay.Default, Nothing)
         Else
            Response.Write("<BR>The DataSet was retrieved from
                the Cache")
          End If

          DataGrid1.DataSource = oDS
          Page.DataBind()
      End Sub
```

You can make caching more intelligent by defining items or keys that the cache depends on. If one of the items or keys is altered, the cache will be automatically cleaned out. However, just like in page caching, you can set the expiration time for the cache. Set the *Cache.Insert* or use the defined parameters of *Cache.NoAbsoluteExpiration* or *Cache.NoSlidingExpiration* to prevent the removal of the cached information.

In data caching, maintaining the freshness of the data is critical. The freshness of data affects whether or not you're working with correct data. The first step is determining how the data will be used and deciding upon the life of the data needed. You can programmatically control the addition and removal of records into the cache by using the *cache.add* and *cache.remove* methods, respectively. However, the *remove* method is not the norm. Usually, items will be removed because the life of the data has expired or scavenging of the cache has occurred. Scavenging is the cache's way of regaining memory. It looks for low priority items and removes them as necessary. Now, don't think that data will just disappear on you. Remember that if you use any of the following methods while adding the data, the corresponding item in the cache will never be removed:

- *Cache.NoAbsoluteExpiration*
- *Cache.NoSlidingExpiration*
- *CacheItemPriority.NotRemovable*
- *CacheItemPriorityDecay.Never*

Developing & Deploying…

Understanding Caching

The following are caching guidelines:

- **Try not to overuse caching.** There is a cost to memory for every item that you cache. Avoid caching items that will rarely be used again or items that can be easily recomputed.
- **Avoid caching items that expire quickly.** Again, there is a high cost to memory and processing if the cache is repeatedly refreshed. Look for items that can stay around for a while.

Caching when used correctly can significantly increase performance of any Web application. It can significantly reduce the number of server round trips taken, thereby reducing overall application performance and overhead.

Tracing in ASP.NET

Everyone has had to implement some form of tracing during some point of writing any Web application. Even from the earliest days of ASP, we have used *Response.Write* to tell us what lines of code are being hit within our programs. However, *Response.Write* can be painful to use. For instance, you never know where the *Response.Write* is going to hit on the page, and having the statements intermingled throughout all the code can get very messy.

ASP.NET provides an excellent integrated tool called *tracing*. Tracing allows you to track step by step what is happening during execution of your applications. The tracing provided by ASP.NET even gives you choices on how you would like to see your tracing output. Output for tracing can be at a page or at application level.

Page-Level Tracing

Page-level tracing will attach a mess of information regarding a single Web page at the end of the page after executing. This is done by adding one simple line of code to the top of your ASP.NET page within the *@Page* directive. The line would look something like this:

```
<%@ Page Language=VB Trace="True" %@>
```

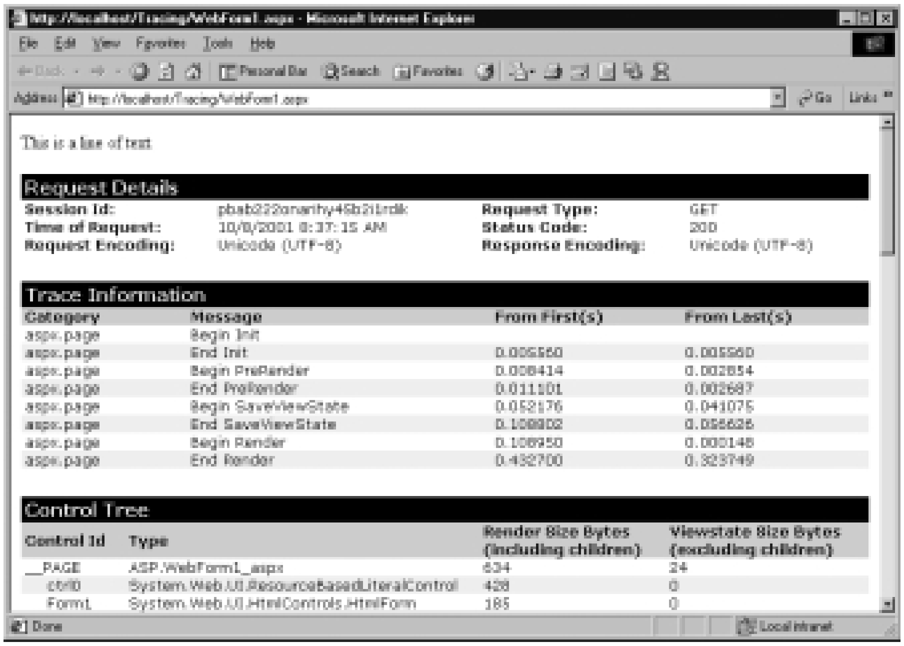
Just so you can see how much is generated for a simple page. Figure 3.42 shows some simple HTML that we placed on an ASP.NET page and has included the *trace* statement.

Figure 3.42 *Trace* Statement on the *@Page* Directive

```
<%@ Page Language="vb" AutoEventWireup="false"
       codebehind="WebForm1.aspx.vb"
       Inherits="Tracing.WebForm1"%>
<%@ Page Trace="true"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
    <head>
     <title></title>
    <meta name="GENERATOR" content="Microsoft Visual Studio.NET 7.0">
    <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
    <meta name="vs_defaultClientScript"          content="JavaScript">
     <meta name="vs_targetSchema"
            content="http://schemas.microsoft.com/intellisense/ie5">
    </head>
    <body MS_POSITIONING="GridLayout">
    <form id="Form1" method="post" runat="server">
       This is a line of text.
    </form>
    </body>
</html>
```

Executing the page as is will produce only the line of text. However, Figures 3.43 through 3.46 show what is displayed when *trace* is turned on for the page.

Figure 3.43 Generated Trace, Page 1



This is a line of text

Request Details

Session Id:	pbab222onanhy45b21rdk	**Request Type:**	GET
Time of Request:	10/8/2001 8:37:15 AM	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

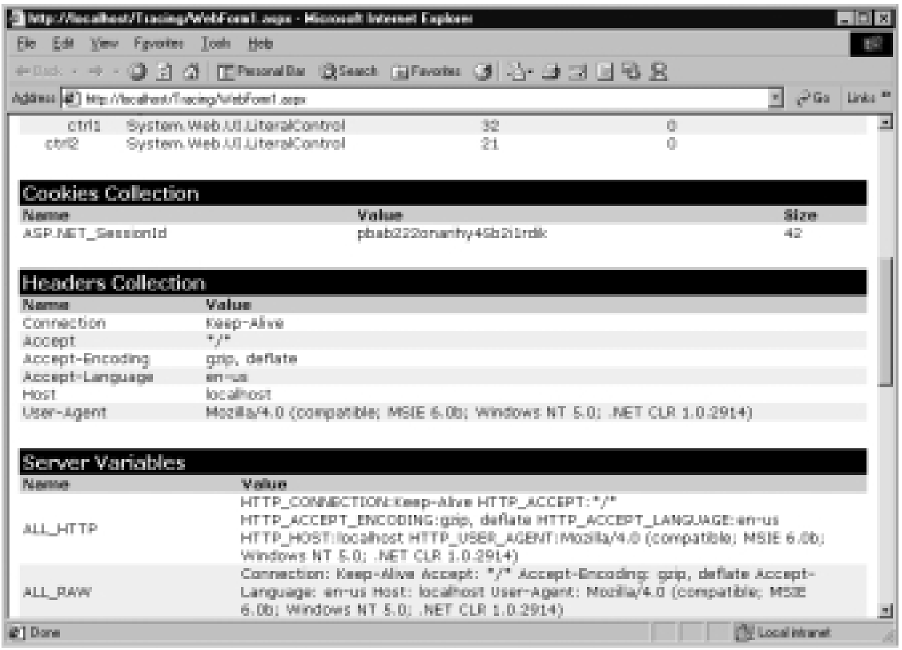
Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin Init		
aspx.page	End Init	0.005560	0.005560
aspx.page	Begin PreRender	0.008414	0.002854
aspx.page	End PreRender	0.011101	0.002687
aspx.page	Begin SaveViewState	0.052176	0.041075
aspx.page	End SaveViewState	0.108802	0.056626
aspx.page	Begin Render	0.108950	0.000148
aspx.page	End Render	0.432700	0.323749

Control Tree

Control Id	Type	Render Size Bytes (including children)	Viewstate Size Bytes (excluding children)
__PAGE	ASP.WebForm1_aspx	634	24
ctrl0	System.Web.UI.ResourceBasedLiteralControl	428	0
Form1	System.Web.UI.HtmlControls.HtmlForm	185	0

Figure 3.44 Generated Trace, Page 2



ctrl1	System.Web.UI.LiteralControl	32	0
ctrl2	System.Web.UI.LiteralControl	21	0

Cookies Collection

Name	Value	Size
ASP.NET_SessionId	pbab222onanhy45b21rdk	42

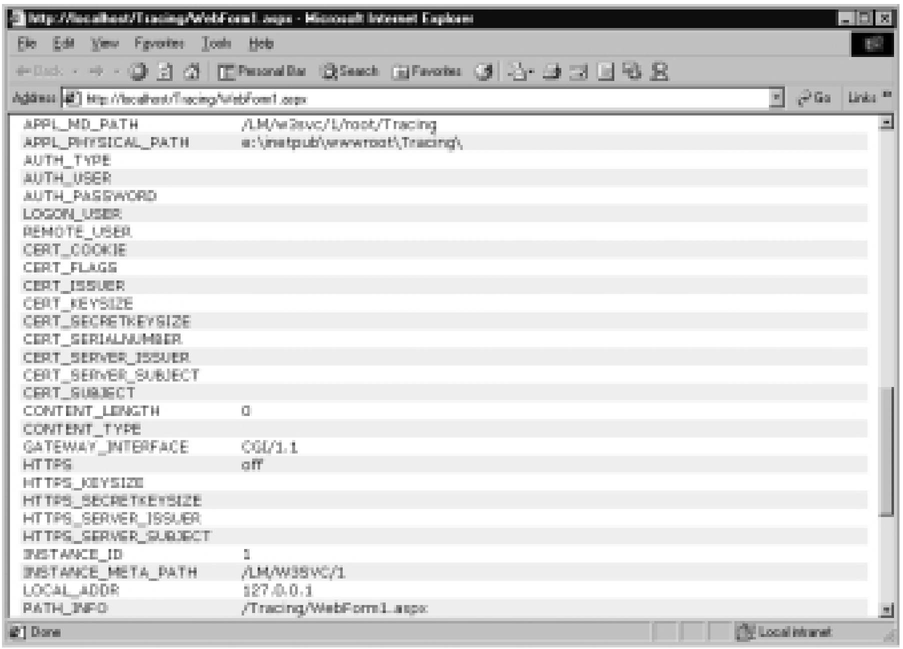
Headers Collection

Name	Value
Connection	Keep-Alive
Accept	*/*
Accept-Encoding	gzip, deflate
Accept-Language	en-us
Host	localhost
User-Agent	Mozilla/4.0 (compatible; MSIE 6.0b; Windows NT 5.0; .NET CLR 1.0.2914)

Server Variables

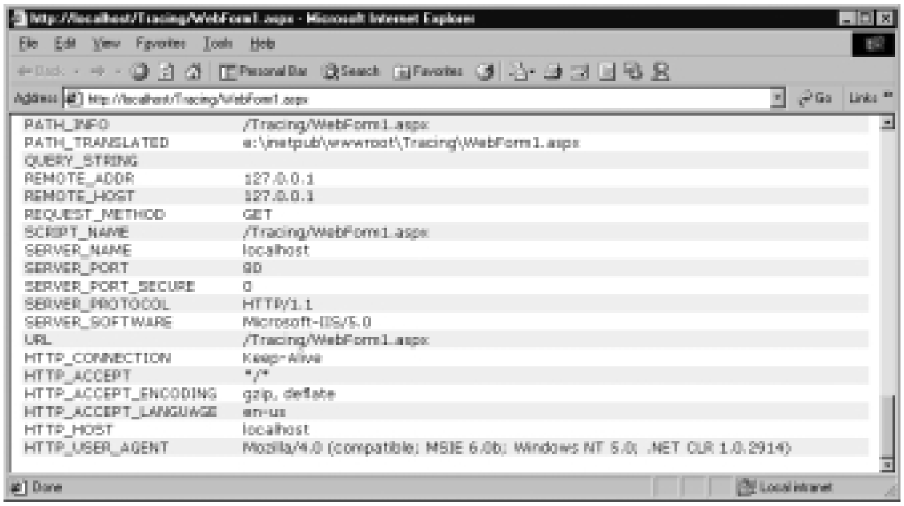
Name	Value
ALL_HTTP	HTTP_CONNECTION:Keep-Alive HTTP_ACCEPT:*/* HTTP_ACCEPT_ENCODING:gzip, deflate HTTP_ACCEPT_LANGUAGE:en-us HTTP_HOST:localhost HTTP_USER_AGENT:Mozilla/4.0 (compatible; MSIE 6.0b; Windows NT 5.0; .NET CLR 1.0.2914)
ALL_RAW	Connection: Keep-Alive Accept: */* Accept-Encoding: gzip, deflate Accept-Language: en-us Host: localhost User-Agent: Mozilla/4.0 (compatible; MSIE 6.0b; Windows NT 5.0; .NET CLR 1.0.2914)

Figure 3.45 Generated Trace, Page 3



APPL_MD_PATH	/LM/w3svc/1/root/Tracing
APPL_PHYSICAL_PATH	e:\inetpub\wwwroot\Tracing\
AUTH_TYPE	
AUTH_USER	
AUTH_PASSWORD	
LOGON_USER	
REMOTE_USER	
CERT_COOKIE	
CERT_FLAGS	
CERT_ISSUER	
CERT_KEYSIZE	
CERT_SECRETKEYSIZE	
CERT_SERIALNUMBER	
CERT_SERVER_ISSUER	
CERT_SERVER_SUBJECT	
CERT_SUBJECT	
CONTENT_LENGTH	0
CONTENT_TYPE	
GATEWAY_INTERFACE	CGI/1.1
HTTPS	off
HTTPS_KEYSIZE	
HTTPS_SECRETKEYSIZE	
HTTPS_SERVER_ISSUER	
HTTPS_SERVER_SUBJECT	
INSTANCE_ID	1
INSTANCE_META_PATH	/LM/W3SVC/1
LOCAL_ADDR	127.0.0.1
PATH_INFO	/Tracing/WebForm1.aspx

Figure 3.46 Generated Trace, Page 4



PATH_INFO	/Tracing/WebForm1.aspx
PATH_TRANSLATED	e:\inetpub\wwwroot\Tracing\WebForm1.aspx
QUERY_STRING	
REMOTE_ADDR	127.0.0.1
REMOTE_HOST	127.0.0.1
REQUEST_METHOD	GET
SCRIPT_NAME	/Tracing/WebForm1.aspx
SERVER_NAME	localhost
SERVER_PORT	80
SERVER_PORT_SECURE	0
SERVER_PROTOCOL	HTTP/1.1
SERVER_SOFTWARE	Microsoft-IIS/5.0
URL	/Tracing/WebForm1.aspx
HTTP_CONNECTION	Keep-Alive
HTTP_ACCEPT	*/*
HTTP_ACCEPT_ENCODING	gzip, deflate
HTTP_ACCEPT_LANGUAGE	en-us
HTTP_HOST	localhost
HTTP_USER_AGENT	Mozilla/4.0 (compatible; MSIE 6.0b; Windows NT 5.0; .NET CLR 1.0.2914)

Even though only one simple line of text was included on the page, all the information you see was generated. The *Trace* summary page breaks up the information into different sections for easier reading. Notice that all of the following sections will not appear unless the code or actions on the page warrant their use. The possible sections that can be displayed are shown in Table 3.4.

Table 3.4 Trace Summary Page Sections

Section	Description
Request Details	This is information about the actual request for the page including items such as Request Type, the time the request came in, and the type of encoding used on the page.
Trace Information	This lists the actions or steps taken during execution of the request. It will list the action along with the time it took to take the action and the time between actions. Categories are provided to group the type of actions. In this case, the simple text we created had only one category.
Control Tree	This is a list of all the controls on the page in hierarchical order. It will also give a size description of the control. Notice that in our simple Web page we had only the page itself and a literal control.
Session State	This is any item held in Session state. In our example, we didn't have any, so the section was left off.
Application State	This is any item held in Application state. Again, we didn't have any items, so the section was ignored.
Cookies Collection	This will list any cookies for the page and their associated value.
Headers Collection	This provides a list of the HTTP headers and their associated values.
Forms Collection	This is a list of forms and their contents.
QueryString Collection	This is a list of QueryStrings and their values.
Server Variables	This is a complete list of server variables and their values.

A great feature of the new ASP.NET tracing facility is the ability to add your own messages. Any messages that you create will be inserted within the generated text on the trace summary report. You can insert two types of messages: *Write* messages and *Warn* messages. Both are methods of the *TraceContext* object on the page.

Trace.Write is for inserting an informational text message within the trace summary. It looks no different than the trace information, except that it has your verbiage on it.

Trace.Warn is also for inserting an informational text message within the trace summary, but it is displayed in red. We prefer *Trace.Warn* messages because they are easier to see in the trace summary.

Both methods are overloaded. Overloading refers to a method that requires different input parameters depending on the format selected. For instance, you can choose to write a *Trace.Warn* with a text message only. Or you can choose to add a category along with the text message. The overloaded formats are as follows for both *Trace.Write* and *Trace.Warn*:

- *Trace.Write (message as string)*
- *Trace.Write (category as string, message as string)*
- *Trace.Write (category as string, message as string, errorInfo as System.Exception)*

Notice the following examples both insert the same text message as defined, but the *Trace.Warn* is displayed in red. The page load event in Figure 3.47 contains two *trace* statements.

Figure 3.47 *Trace* Statements

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles MyBase.Load

     Trace.Write("Trace.Write - we're in the Page Load")
     Trace.Warn("Trace.Warn - we're in the Page Load"

    End Sub
```

Now let's take a look at how it gets generated (see Figure 3.48).

Notice that the *Trace.Write* is exactly the same as the *Trace.Warn* except for the color. This is why we prefer using the *Trace.Warn* because it stands out and is much easier to read. We could have created our own category for the messages (notice that the Category column is blank) by using the overloaded message to add one.

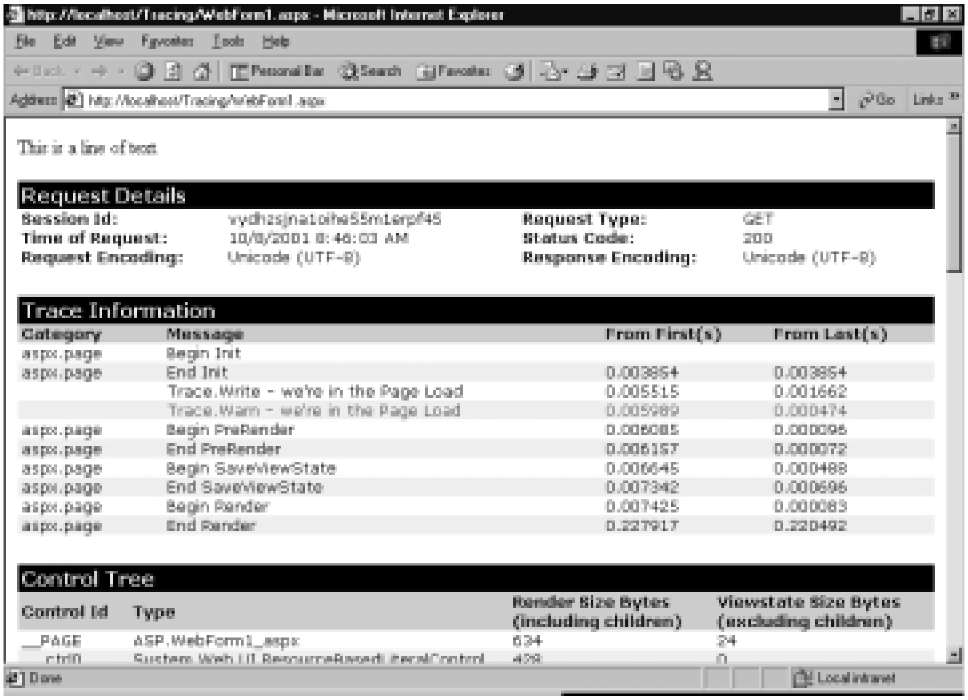


Figure 3.48 Generated Page with Attached *Trace* Page

http://localhost/Tracing/WebForm1.aspx - Microsoft Internet Explorer

Address: http://localhost/Tracing/WebForm1.aspx

This is a line of text.

Request Details

Session Id:	vydhzsjna1oihe55m1erpf45	**Request Type:**	GET
Time of Request:	10/8/2001 8:46:03 AM	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin Init		
aspx.page	End Init	0.003854	0.003854
	Trace.Write - we're in the Page Load	0.005515	0.001662
	Trace.Warn - we're in the Page Load	0.005989	0.000474
aspx.page	Begin PreRender	0.006085	0.000096
aspx.page	End PreRender	0.006157	0.000072
aspx.page	Begin SaveViewState	0.006645	0.000488
aspx.page	End SaveViewState	0.007342	0.000696
aspx.page	Begin Render	0.007425	0.000083
aspx.page	End Render	0.227917	0.220492

Control Tree

Control Id	Type	Render Size Bytes (including children)	Viewstate Size Bytes (excluding children)
__PAGE	ASP.WebForm1_aspx	634	24
ctrl0	System.Web.UI.ResourceBasedLiteralControl	428	0

Done | Local intranet

For further tracing abilities, the *TraceContext* object of the page has been defined with two available properties. You can set these properties at the page level or code them to turn tracing on and off during application execution:

- **IsEnabled** This refers to the page-level tracing and tells whether or not tracing is on. This property is read/write, and you can code against it. You could dynamically turn the tracing on and off when you need.
- **TraceMode** This enables you to decide how the information in the *trace* summary should be presented. The choices are *SortByCategory* or *SortByTime*. This is also a read/write property, and you can set it through code.

Application-Level Tracing

This is just like page-level tracing, but for an entire application. This type of tracing will send the summary information to a log file that you can retrieve after execution of the application. You can set the maximum number of logs to retain for an application. This means that you can create a history of actions and retrieve them when you need them. Application-level tracing even has a *localOnly* setting that will allow you to track user actions being performed on remote clients. The *localOnly* setting can be beneficial during testing cycles when you want to monitor how the application is used.

NOTE

Although you can set the *localOnly* setting to operate in production, you should not do this lightly. The overhead on the application, and possibly the network, can be detrimental to performance, not to mention expensive.

In order to add application-level tracing, you must update the Web.config file for the application. The Web.config file is located in the root directory of the application. Open the Web.config file either through Visual Studio.NET or in Notepad and locate the <system.Web> section. Add a *Trace* element to this section that will turn the tracing on.

NOTE

Remember that Web.config is an XML file and that anything you add should be well-formed.

The entry into Web.config should look something like the following, depending on the attributes you select:

```
<system.Web>


<trace enabled = 'true'></trace>
</system.Web>
```

Table 3.5 shows the attributes of the *Trace* element.

Table 3.5 Attributes of the *Trace* Element

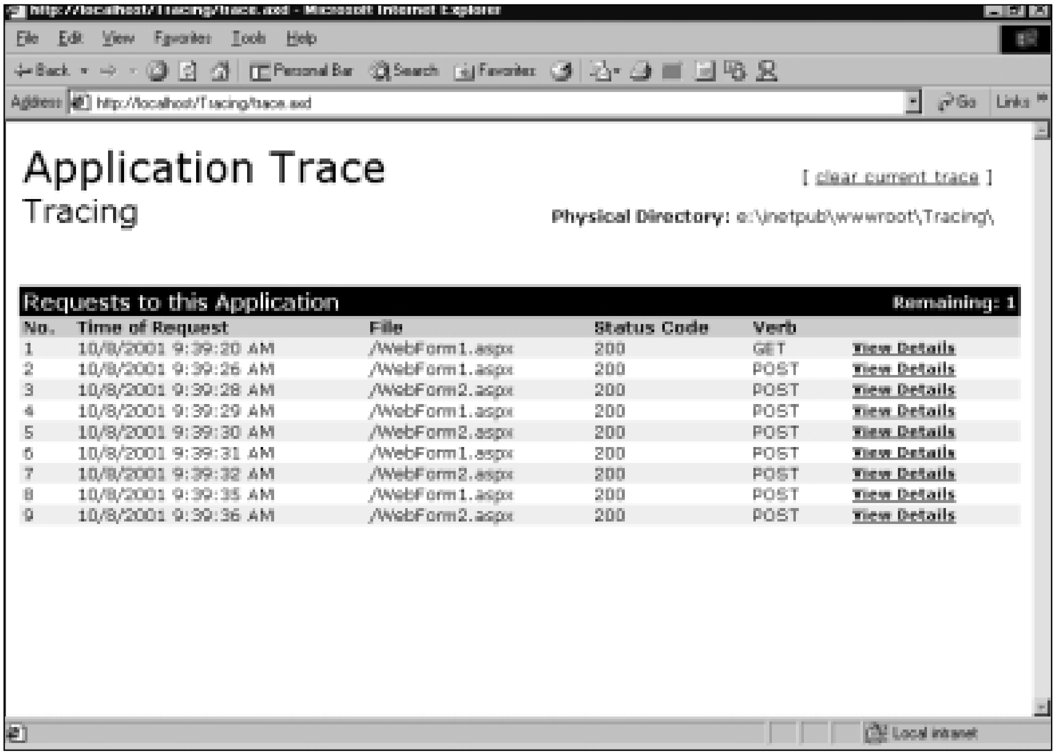
Element	Default	Description
Enabled	False	This will turn the application-level tracing on and off.
RequestLimit	10	This is the number of HTTP requests for which to capture and store tracing information. The log will display up to that number at any given time and will add new ones, deleting the oldest one as the limit is reached.
PageOutput	False	This is to display tracing information at the end of each page. Note that the trace information is always collected regardless of whether it is turned on or not.
TraceMode	SortByTime	Just as you saw in page-level tracing, you can sort the results. The default is *SortByTime*, but you can also set it to *SortByCategory*.
LocalOnly	*True*	This parameter will enable the log to be seen by local clients or by remote clients as well. This is great if you want to see tracing information for another computer/user's actions.

The trace logs will be located in the trace.axd file in the root directory of your application. However, you won't be able to physically navigate to a log file in the root directory, because the logs are part of a special URL that is created when trace is turned on and is hidden. The trace.axd summary page will display the list of generated logs. The list will contain up to the maximum as defined by the *RequestLimit* element. To view one of the log files, click on the *ViewDetails* link of the request to see the trace information.

Figure 3.49 shows what happens when tracing is turned on in a Web.config file and a simple Web page is executed. The list of logs gets generated for each request to the page.

You can select the tracing log you would like to view by clicking the appropriate **View Details** link. The tracing screen will open up and present what was occurring in the application during that request. As you can see, the tracing facility provided for you in ASP.NET can be a powerful tool in the debugging process of your Web applications and can save you the heartache of typing *Response.Write* statements throughout your code.

Figure 3.49 Application Trace Log


http://localhost/Tracing/trace.axd - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Personal Bar Search Favorites
Address http://localhost/Tracing/trace.axd
Go Links
Local intranet

Application Trace

Tracing

[clear current trace]

Physical Directory: e:\inetpub\wwwroot\Tracing\

Requests to this Application — **Remaining: 1**

No.	Time of Request	File	Status Code	Verb	
1	10/8/2001 9:39:20 AM	/WebForm1.aspx	200	GET	View Details
2	10/8/2001 9:39:26 AM	/WebForm1.aspx	200	POST	View Details
3	10/8/2001 9:39:28 AM	/WebForm2.aspx	200	POST	View Details
4	10/8/2001 9:39:29 AM	/WebForm1.aspx	200	POST	View Details
5	10/8/2001 9:39:30 AM	/WebForm2.aspx	200	POST	View Details
6	10/8/2001 9:39:31 AM	/WebForm1.aspx	200	POST	View Details
7	10/8/2001 9:39:32 AM	/WebForm2.aspx	200	POST	View Details
8	10/8/2001 9:39:35 AM	/WebForm1.aspx	200	POST	View Details
9	10/8/2001 9:39:36 AM	/WebForm2.aspx	200	POST	View Details

Summary

So now you've had a chance to read through some of the aspects of ASP.NET programming and have hopefully seen some of the benefits over its predecessor ASP. Some of those aspects are the flexibility and customization provided by the .NET Framework to which ASP.NET is attached. Although this chapter is not comprehensive, it does provide a sound overview of the new Web application development environment.

By now you should understand that ASP.NET is an extension to the .NET Framework and that it sits on top of the Common Language Runtime, providing an almost unlimited opportunity to customize and extend Web applications. It's one of the premier reasons why ASP was rewritten from the ground up and not just added onto.

The guiding light behind the rewrite was to create a better way to develop Web applications and provide more functionality. Although some traditional ASP methods have gone out with the wind, brand new and better concepts have come in. Some of those include server-side processing of code, a development environment that is integrated with Visual Studio.NET to provide drag-and-drop development, better caching, state management, and debugging support. And these are just the tip of the iceberg.

The new *WebForm* controls provide everything from basic labels on a page to rich controls, such as Calendars and Ad Rotators that you can drag and drop onto a Web page. No longer do you have to write a mountain of script to create a grid with functional processing. The new *DataGrid* Web control is a great example of the robustness provided in the new suite of Web Form controls.

You saw how the development environment is flexible, and you can use it the way you want it to be used. You can choose to write inline code that is similar to how it was done in traditional ASP, or you can separate it with the use of *Codebehind* classes. You can choose to develop in a text editor or use the new Visual Studio.NET environment for ASP.NET.

You saw the use of configuration files and how they can play a substantial role in development of Web applications. The ability to use the XML-based Web.config to assist in defining application-level settings is a huge benefit that was not available in ASP.

We looked at State management and caching facilities that can be an integral part of reducing overhead in applications. You saw that they are more than just holding areas for data, that you can actually programmatically manipulate what is stored, when, and how.

You saw the new tracing facilities that are completely supported in ASP.NET and how they can significantly reduce the use of *Response.Write* statements all over your code. The tracing facilities are easy to implement and will play an integral part in debugging your Web applications. Remember, no matter how you choose to develop or design your Web applications using ASP.NET, keep in mind that everything you do has tradeoffs. Even though the new development environment makes it very easy to get going, it is very important to consider the overall architecture of a Web application. If you want a fast application, you're going to have to consider caching; however, the next decision should be to determine how much is necessary and where it should be used. Use the new tools and functionality provided in ASP.NET, but take the time to determine the best method for the application you are building. Overall, I suggest just giving ASP.NET a try. It's a fun new development framework that provides a lot of flexibility and expandability that wasn't available in traditional ASP development. Your going to quickly come up to speed and wonder how you ever survived coding in ASP.

Solutions Fast Track

A Quick Look at ASP.NET

☑ A Web Form contains two components: code and content. The content component of a Web Form can contain Web Form Server controls. Web Form Server controls contain the following types of controls: HTML Server control, ASP.NET Server control, Validation controls, and User controls.

☑ The content component basically concerns itself with display issues. The code components are the "glue" that binds things up.

What's New in ASP.NET

☑ The new WebForm controls provide server-side processing with an event-driven programming model.

☑ The Web.config is an XML-based file that provides application specific directions.

Migrating from ASP to ASP.NET

☑ Realize that ASP.NET was written from the ground up to fix many of the problems found in the interpreted scripting language of ASP and that many items in ASP won't easily port to ASP.NET.

☑ ASP.NET provides an Inline coding facility for ASP developers who are accustomed to that type of development, but it also provides a code behind class that provides a facility for easily separating content code from design code.

State Management in ASP.NET

☑ Several types of State management are provided for in ASP.NET; they include Application, Session, Cookies, and Out-of-Process. Each has a distinct use and benefit over the other.

Tracing in ASP.NET

☑ Tracing has been built into ASP.NET to provide an easy mechanism to pinpoint application functionality.

☑ Tracing in ASP.NET reduces or eliminates the need to place *Response.Write* statement throughout development code.

Frequently Asked Questions

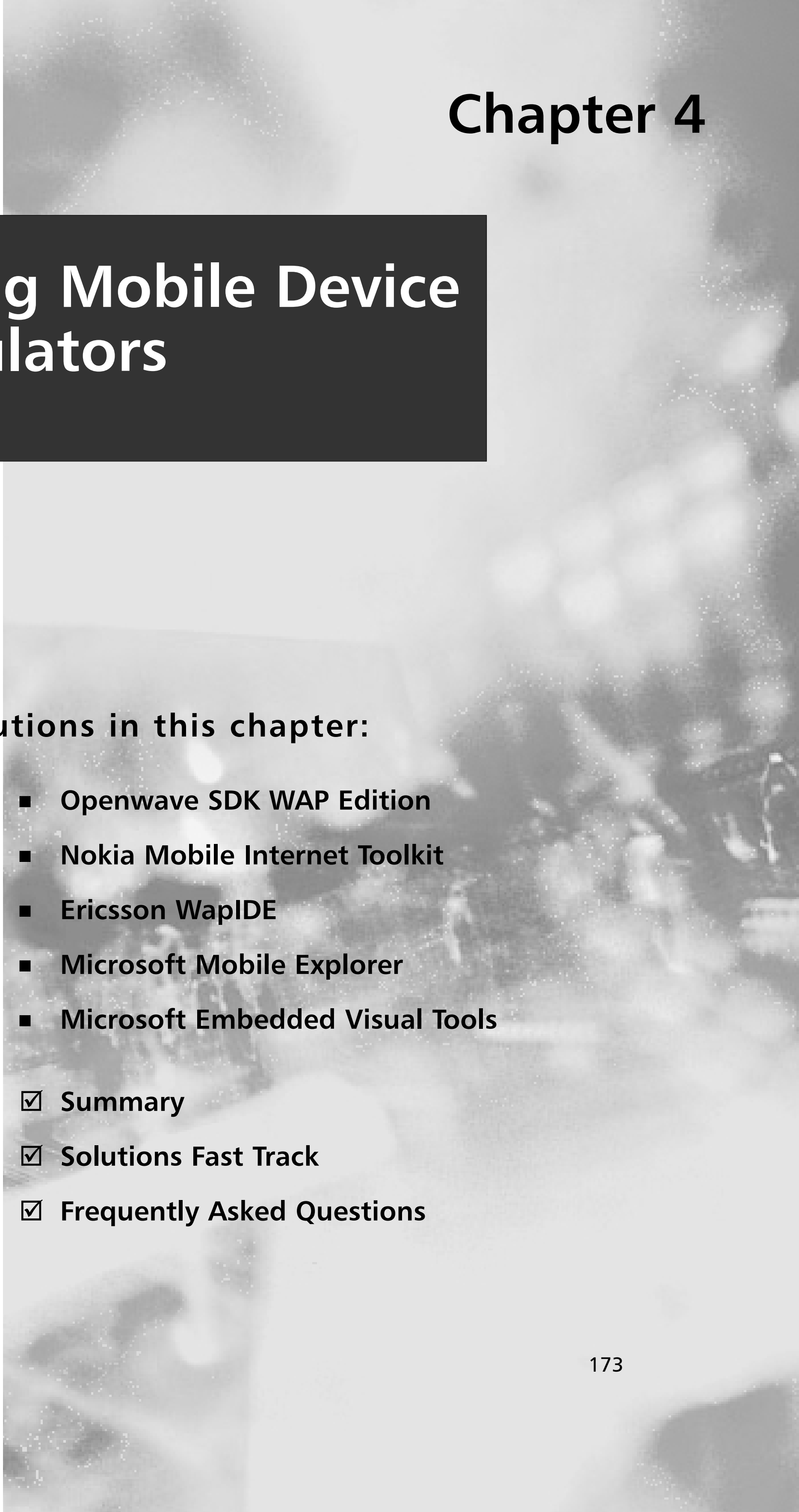
The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: Can a page have a mix of .NET languages within it?

A: No, the page and the project it resides in should all be created within the same .NET language. However, referenced components used within the project can be of any .NET language. This is because all code gets compiled down to the same Microsoft Intermediate Language (MSIL). At this level, the code is all the same language.

Q: Is there a better or worse way for transferring control between pages?

A: Yes, two methods are available, depending on what is needed: *Response.Redirect* and *Server.Transfer*. The *Response.Redirect* actually has the page perform a redirect after essentially checking with the client's browser to make sure they really want to go somewhere else. The *Server.Transfer* will take you directly to the target without asking. The benefit is that there are less round trips to perform the same action. However, another difference is that *Response.Redirect* will allow you to pass *QueryStrings*, whereas *Server.Transfer* will not. So, unless you have a specific need to pass *QueryStrings*, I recommend the *Server.Transfer* method to reduce application execution overhead.



Chapter 4

Using Mobile Device Emulators

Solutions in this chapter:

- Openwave SDK WAP Edition
- Nokia Mobile Internet Toolkit
- Ericsson WapIDE
- Microsoft Mobile Explorer
- Microsoft Embedded Visual Tools

☑ Summary

☑ Solutions Fast Track

☑ Frequently Asked Questions

Introduction

Development of mobile applications is quite different from the traditional Web application development. In the traditional system, you develop, test, and use the application on almost the same platform. But in the case of mobile applications, you will use the more productive environment of a personal computer for developing and testing your applications while targeting a mobile device execution environment. A mobile device is an altogether different platform for running applications. As a mobile device developer, you need to address the special nature of mobile devices: low processing power, small screen, different methods of input, and the premium cost of bandwidth. Not only this, but you might need to test your application on a variety of devices to ensure correct rendering in all of them.

To increase productivity during mobile device development and to cut costs, you should use emulators instead of real mobile device for most of your development and testing requirements. Emulators allow you to test your application directly from desktop computer, and they do not require an actual device or a wireless connection. Emulators also include additional development tools, such as the ability to view page source or device state. Several emulators are available, mostly available through manufacturers to help developers develop applications targeting their devices. You can install several emulators on your computer and ensure desired results in each of that devices without leaving the environment you are already familiar with.

Developing & Deploying…

Emulator versus Real Device

Emulator software are your best friends for wireless application development because they provide productive environments with powerful tools for developing, testing and debugging. Also, it is much more economical to install multiple simulators on a developer's machine rather than purchasing real devices for each of the developers in a development team.

But you must remember that an emulator will only *emulate* the behavior of a real device. Although manufacturers strive to ensure that the behavior of their emulator closely mimics that of a real mobile device, there will inevitably be differences.

You also need to bear in mind that although your application may seem simple enough to use on an emulator, you may have made design decisions that render it unusable on a real device. *Always* perform your final testing on as many of the "real" target devices as you can get your hands on. This chapter lays the foundation for developing and testing mobile applications using emulators—it also includes information on installing and using emulators from all of the major browser vendors. We mainly focus on the following popular mobile device emulators:

- Openwave SDK WAP Edition
- Nokia Mobile Internet Toolkit
- Ericsson WAP IDE
- Microsoft Mobile Explorer
- Microsoft Embedded Visual Tools

But even if you are using an emulator that is not in this list, the basic behavior is going to be the same.

Developing & Deploying…

Mini-Browser Market Watch

In the United States, Openwave mini-browsers dominate the wireless handsets, whereas in Europe, Nokia has the largest share of Gateways and micro-browsers. It's good to know the market scenario, but some developers build their application around only one type of browsers while ignoring others. In European markets, for example, you would find applications working properly in the Nokia browsers but performing poorly in Openwave, and vice versa for American markets.

This is not a good design from a long-term strategic point of view. Mergers, alliances, and changes in market situations will require that your application performs well on other significant browsers also. If you perform thorough testing of your application with multiple browsers at development level (possibly all the browsers mentioned in this chapter), it will take significantly less effort in making changes for compatibility with other devices as compared to making changes when your application is mature.

Openwave SDK WAP Edition

The Openwave SDK WAP Edition 5.0 (former versions known as UP.SDK) is an all-new version of this popular simulator. Openwave and its previous edition combined together are the most popular simulating environments with the mobile developers. It has support for a wide variety of handsets including Alcatel, Ericsson, Motorola, and Openwave using Openwave Mobile Browser. (For a complete list of phones supporting Openwave Mobile Browser, please refer to http://developer.openwave.com/resources/phones.html.)

Unlike the old versions, this new version comes with an integrated development environment (IDE) for both creating and testing mobile applications. The text editor included with IDE supports syntax highlighting for WML and WMLScript content. The Openwave SDK is shipped with support of the following standards:

- **Download Fun (a part of GSMA M-Services initiative)** Download Fun is a content delivery solution for Communications Service Providers (CSP). You can read more about it at www.openwave.com/products/platform_services/download_fun. The M-Services initiative from the GSMA contains a requirement for handsets to support a GUI.
- **WML 1.3 with GUI extensions** This is a WAPForum specification; you can read more about it at the WAPForum Web site at www.wapforum.org.
- **WAP Push 1.2.1 and WAP 1.2** These are again WAPForum specifications; you can read more about it at the WAPForum Web site at www.wapforum.org.

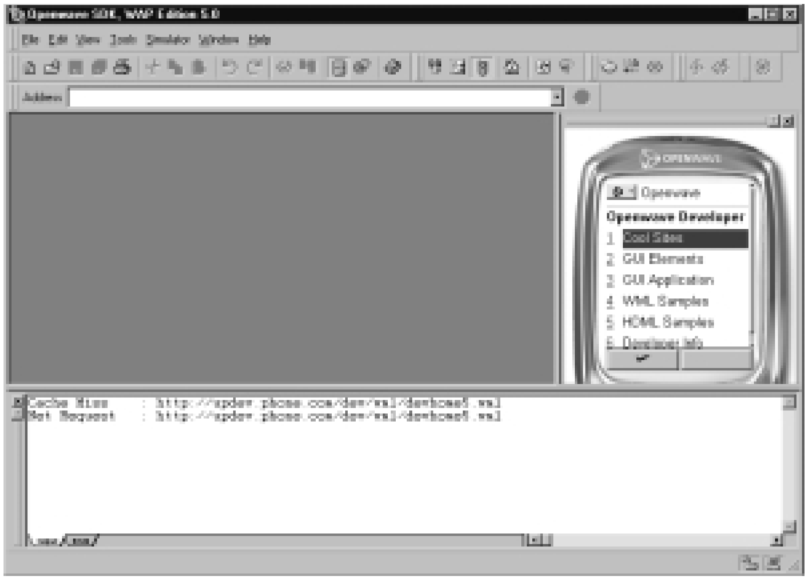
Installing Openwave SDK WAP Edition

You can download the Openwave SDK from the Openwave developer's Web site: http://developer.openwave.com. You have to register before you will be allowed to download the SDK. Two different editions of SDK are available, one is for Windows NT/2000 operating systems and the other one is for Windows 98/ME operating systems.

No specific system requirements exist for running Openwave SDK, other than the correct operating system, which makes installation of Openwave easy and quick. It also does not require additional installation of Java runtime as required by some other simulators. Openwave recommends using Windows 2000 (Service Pack 2) for optimum IDE performance.

After you have completed installation, select **Start | Programs | Openwave SDK, WAP Edition 5.0 | Openwave SDK, WAP Edition 5.0** to open up the IDE, as seen in Figure 4.1.

Figure 4.1 Openwave SDK, WAP Edition 5.0 IDE (Image courtesy of Openwave Systems, Inc.)



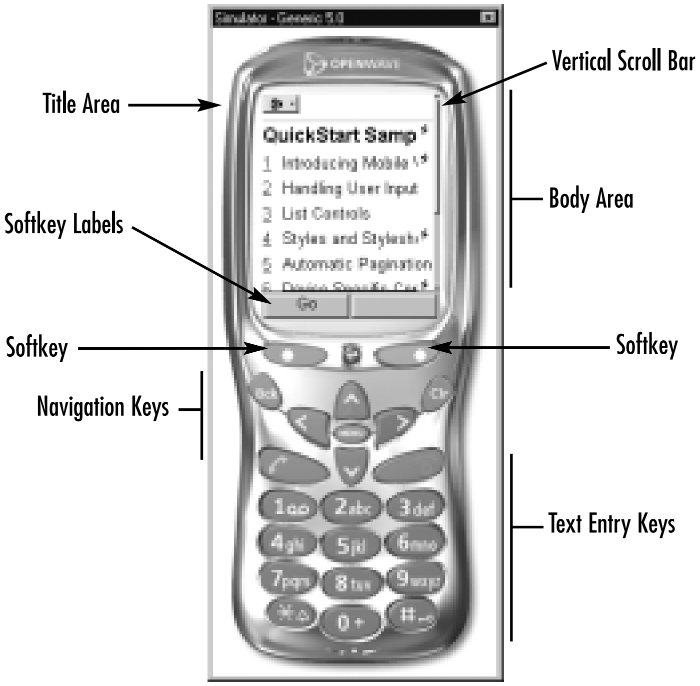
```
Cache Miss    : http://updev.phone.com/dev/wml/devhome5.wml
Net Request   : http://updev.phone.com/dev/wml/devhome5.wml
```

The initial output in the phone is coming from http://updev.phone.com/dev/wml/devhome5.wml. You must be connected to Internet to see this initial output; if you are not connected, you will get an HTTP error because this URL can not be reached, but you should be able to request the files locally stored on your computer or network. If you have Microsoft Mobile Internet toolkit installed on your computer, you can type the following URL in the Address bar of the IDE: **http://localhost/MobileQuickStart/Samples/default.aspx**, which will show you the Mobile QuickStart documentation.

Using Openwave SDK WAP Edition

Figure 4.2 shows the layout of the simulator. The use of different keys has been marked. You will find that most phones have similar key and features, but some phones might be different—refer to the documentation of that particular phone to learn about its usage.

Figure 4.2 Openwave Generic Simulator (Image courtesy of Openwave Systems, Inc.)

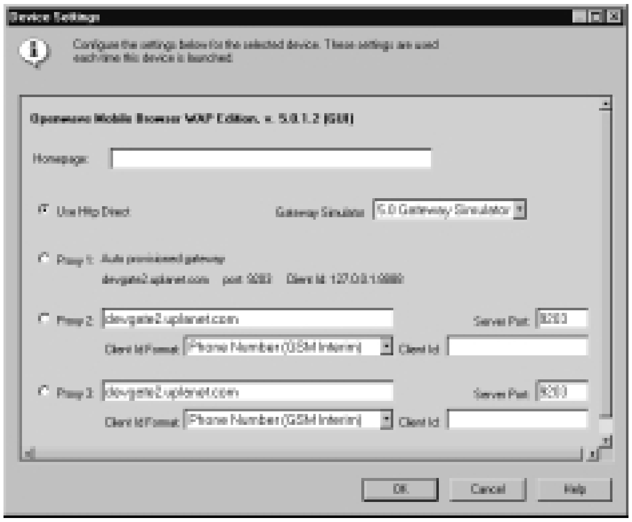


To visit a new Web site, you can either type the URL in the address bar of the IDE or you can select **Simulator | Go To Address** from the IDE menu. Once you have a Web Page displayed, you can navigate using the navigation keys. If any messages are displayed on the Softkey labels, you can select them by pressing specific Softkey. In the real device, you have to enter text using the keypad, but in this rich simulation environment, you can also use your keyboard for convenient data entry.

The default skin for the simulator is Generic 5.0; to change it, select **Simulator | Select Device**. Selecting **Simulator | Device Settings** will allow you to set your home page and connectivity option for the simulator (see Figure 4.3).

The default connectivity option is **Use Http Direct**, this option uses the internal WAP Gateway to connect to your URLs. For most of your development process, you should connect through this internal gateway. Towards the later stages of development and testing, if you would like to test your application using a real WAP Gateway, you can specify the address of your WAP Gateway in the Proxy options.

Figure 4.3 Simulator Device Settings (Image courtesy of Openwave Systems, Inc.)



Device Settings

Configure the settings below for the selected device. These settings are used each time this device is launched.

Openwave Mobile Browser WAP Edition, v. 5.0.1.2 (GSR)

Homepage:

Use Http Direct — Gateway Simulator: 5.0 Gateway Simulator

Proxy 1: Auto provisioned gateway
devgate2.uplanet.com port: 9203 Client Id: 127.0.0.1:9999

Proxy 2: devgate2.uplanet.com — Server Port: 9203
Client Id Format: Phone Number (GSM Interim) — Client Id:

Proxy 3: devgate2.uplanet.com — Server Port: 9203
Client Id Format: Phone Number (GSM Interim) — Client Id:

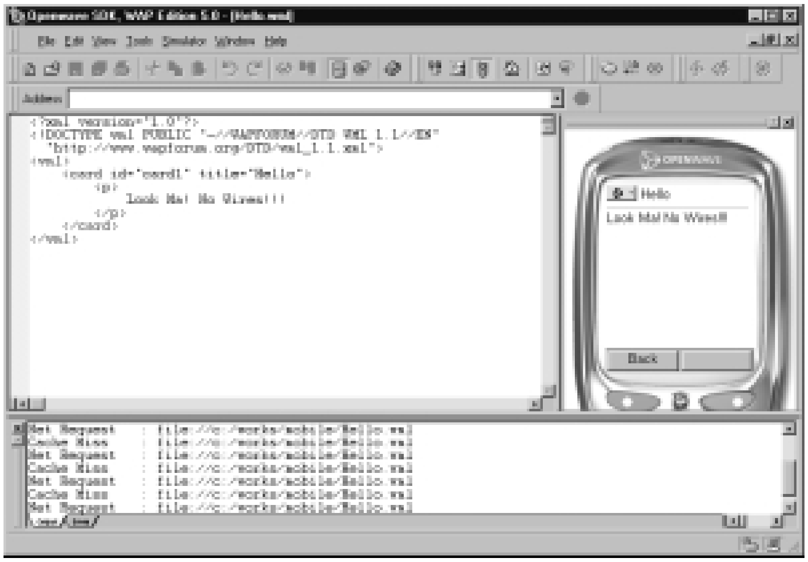
You would specifically use this option to get the feel of performance of your application in real-life scenarios and to test gateway-specific features, such as cookie support and so on. Using a real WAP gateway is a good test for user experience because the response time from a real WAP gateway will also fluctuate due to traffic on the gateway. One of your tests could generate a result in 5 seconds, and another test could generate a result in 15 seconds. This can help you pinpoint critical areas in your application where you need to improve performance.

Using Openwave SDK IDE for Development

Though you can develop your wireless application in any environment and test them using Openwave Simulator, The SDK IDE provides you with several useful features for writing Wireless Markup Language (WML) and WMLScript code including syntax coloring and highlighting. Using the built-in text editor is similar to using any typical text editor. A typical development session on Openwave SDK IDE may look like Figure 4.4. You can preview your program in the Openwave browser by clicking on **File | Preview**.

Some useful tools that you will use while developing applications are listed in Table 4.1.

Figure 4.4 Openwave SDK Development Environment (Image courtesy of Openwave Systems, Inc.)


Openwave SDK, WAP Edition 5.0 - [Hello.wml]
File Edit View Tools Simulator Window Help
Address
Hello
Look Ma! No Wires!!!
Back

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
  "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
    <card id="card1" title="Hello">
        <p>
            Look Ma! No Wires!!!
        </p>
    </card>
</wml>
```

```
Net Request    : file://c:/works/mobile/Hello.wml
Cache Miss     : file://c:/works/mobile/Hello.wml
Net Request    : file://c:/works/mobile/Hello.wml
Cache Miss     : file://c:/works/mobile/Hello.wml
Net Request    : file://c:/works/mobile/Hello.wml
Cache Miss     : file://c:/works/mobile/Hello.wml
Net Request    : file://c:/works/mobile/Hello.wml
```

Table 4.1 Openwave SDK Useful Windows for Development and Debugging

Tool	Description
Browser output window	The Browser Output Window is open by default when you first open the SDK, You can show or hide this window by selecting **View | Browser Output Window** from the menu. This window provides useful information regarding from where the page is loading. You can see if the page is actually coming from an Internet location or from the simulator cache.
HTTP response window	This window will help you view HTTP Response as either plain text or as markup, by selecting the appropriate tab at the bottom of this window. You can show or hide the window by selecting **View | HTTP Response Window**.
Debug windows	Using the integrated debugger, you can inspect session variables, cookies and navigation history. You can show or hide the windows by selecting them from **View | Debug Windows**. If you wish to clear cache and all debug information, you can select **Simulator | New Session**.

These windows will provide you very useful information about how a program is being executed and will help you track the errors in your program. In the previous illustrations, you have seen how to write and test your program using the Openwave SDK Integrated Development Environment. You don't need to write a program in IDE to test it using Openwave Browser. You can write your programs anywhere and then open them in the Openwave Browser by typing the file path. If you wish you can also save your program files to a Web Server, if you are using Windows 2000, doing this can be as simple as copying the WML file (or your ASPX file that generates WML content) to the c:\inetpub\wwwroot directory or any other directory inside it. You will then request the file in Openwave browser using HTTP protocol by framing your URL something like http://localhost/hello.wml.

Nokia Mobile Internet Toolkit

The Nokia Mobile Internet Toolkit is a complete suite of tools for wireless application development. It provides facilities for editing, running, and debugging your WAP applications. Using different simulators, you can also test your output for specific Nokia handsets. At the time of this writing, version 3.0 is the latest available version of the Nokia Mobile Internet Toolkit. It supports the various standards summarized in Table 4.2.

Table 4.2 Nokia Mobile Internet Toolkit Standards Support

Component	Standard Supported
Nokia Mobile Browser Simulator	XHTML Basic, CSS Mobile Profile, WAP June 2000
WAP June 2000 Simulator	WAP June 2000, MeT CUE 0.4, Location Info
Server Simulator	WAP June 2000
Nokia 6210 Phone Simulator	WAP Version 1.1
Nokia 7110 Phone Simulator	WAP Version 1.1

Installing Nokia Mobile Internet Toolkit

The Nokia Mobile Internet Toolkit is freely downloadable from http://forum.nokia.com, although you are required to register in the forum before you can download. Once you register, you will have a unique login and password that you will use to log on to the forum to access the download area. You can reach the download area by following links to **Technologies | WAP | Nokia Mobile Internet Toolkit** from the home page. In addition to the toolkit, you will also find individual links to download Nokia Mobile Internet Toolkit Simulators. These will help you run and test your applications for specific Nokia Handsets.

The Nokia Mobile Internet Toolkit 3.0 is shipped as a Zip file. You will need to unzip the contents of this file to a temporary directory before running the installation executable setup.exe. Once the installer is running, simply use the default settings, unless you want to install to a different path.

System Requirements

The Nokia Mobile Internet Toolkit version 3.0 has the following minimum requirements:

- Pentium class CPU 300 MHz or faster
- 128MB RAM
- 50MB HDD
- Windows NT4 (SP4 or higher); or Windows 98; or Windows 2000 (SP2 or higher)
- Java Runtime Environment 1.3.1 or higher, which you can download from www.javasoft.com.

Installing Additional Simulators

The default installation of the Nokia Mobile Internet Toolkit 3.0 includes two device profiles:

- A generic device type that is compatible with the June 2000 WAP specification
- The Nokia Mobile Browser Simulator, which supports emerging standards such as XHTML and WAP CSS, as well as the June 2000 WAP specification

Note that neither of these two browsers is actually installed in a phone, nevertheless, they make a good base to begin your testing because they are designed with standards compliance in mind—that is, sites that work well with the generic browsers *should* also work well with all other standards-compliant browsers.

Nokia also supplies a number of device-specific simulators that you can download for use within the toolkit. At the time of writing, three handset simulators are available with this toolkit:

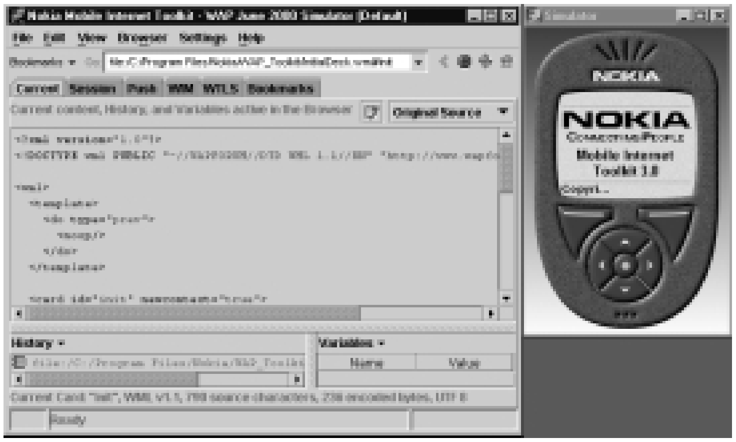
- Nokia 6120 Handset Simulator
- Nokia 7110 Handset Simulator (September 2000 release)
- Nokia 7110 Handset (January 2000 release)

The reason for the two versions of the 7110 simulators is that the WAP browser on the 7110 was changed on some of the later models.

Running the Nokia WAP Toolkit 3.0

After you have completed your installation, you can run the Nokia WAP Toolkit by clicking **Start | Programs | Nokia Mobile Internet Toolkit | Mobile Internet Toolkit**. You will be prompted to register—this is the same registration that you will have needed to complete in order to download the toolkit from http://forum.nokia.com—just click **Already Registered**. Note that the toolkit has an expiration date—this is to ensure that developers use the most up-to-date version. You will now see the default opening screen for this IDE, which will look like Figure 4.5.

Figure 4.5 The Nokia Mobile Internet Toolkit Development Environment



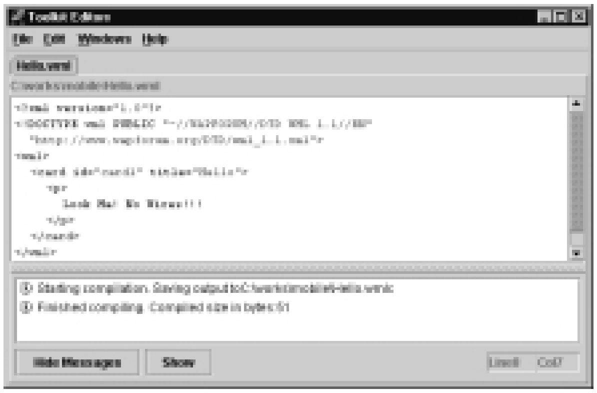
```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "http://www.wapfo

<wml>
  <template>
    <do type="prev">
      <prev/>
    </do>
  </template>

  <card id="init" newcontext="true">
```

File Edit View Browser Settings Help
Bookmarks
Current Session Push WIM WTLS Bookmarks
Current context, History, and Variables active in the Browser
Original Source
History
Variables
Name Value
Ready
NOKIA
NOKIA
Connecting People
Mobile Internet Toolkit 3.0
Copyri...

Using Nokia Mobile Internet Toolkit

You can use Nokia Mobile Internet toolkit for creating and testing WML and WMLScript programs. It supports syntax coloring and highlighting for WML syntax. You can create a new program by clicking **File | New**; this will open Toolkit Editor where you can type your program. Figure 4.6 shows a small program.

Figure 4.6 The Nokia Mobile Internet Toolkit Editor

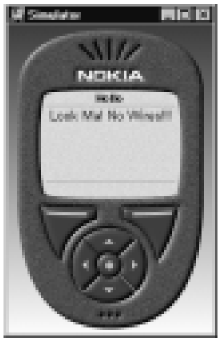


```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
  "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <card id="card1" title="Hello">
    <p>
      Look Ma! No Wires!!!
    </p>
  </card>
</wml>
```

```
Starting compilation. Saving output to C:\work\s\mobile\Hello.wmlc
Finished compiling. Compiled size in bytes:51
```

Once you are done with typing the program, click **Show** to compile the results, which you will be able to see immediately in the simulator. Figure 4.7 shows sample output on the default WAP June 2000 simulator.

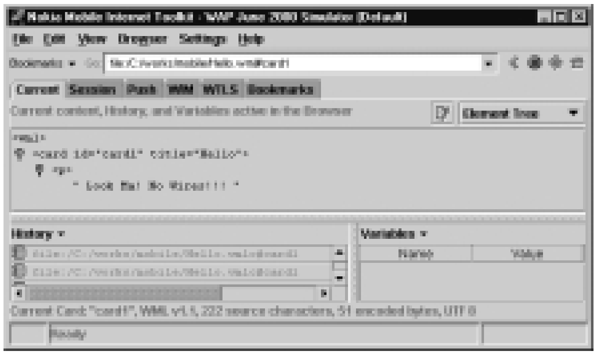
Figure 4.7 The Nokia WAP June 2000 Simulator



While you see this in simulator, you can also look at the source code in the Current tab of the main toolkit window. You also have options to see the element tree and byte code of the WML content from a drop-down list in the same tab as shown in Figure 4.8 (Please refer to sidebar titled "WML Compilation" for more information about byte code).

In addition to this, you can find additional support in the toolkit for debugging purpose. In the Current tab, you can also see the history information and current values of variables. The session-related information belongs to the Session tab, which is shown in Figure 4.9.

Figure 4.8 Current WAP Contents in Element Tree View


Nokia Mobile Internet Toolkit - WAP June 2000 Simulator (Default)
File Edit View Browser Settings Help
Bookmarks Go
Current Session Push WIM WTLS Bookmarks
Element Tree
History
Variables
Name Value
Ready

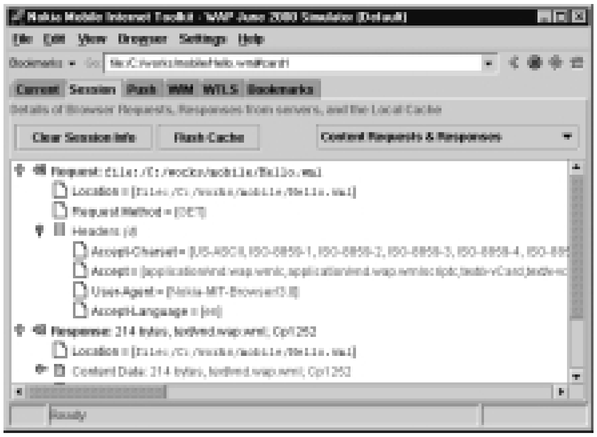
Current content, History, and Variables active in the Browser

```
<wml>
  <card id="card1" title="Hello">
    <p>
      " Look Ma! No Wires!!! "
```

file:/C:/works/mobile/Hello.wml#card1
file:/C:/works/mobile/Hello.wml#card1

Current Card: "card1", WML v1.1, 222 source characters, 51 encoded bytes, UTF 8

Figure 4.9 Current WAP Contents in Element Tree View


Nokia Mobile Internet Toolkit - WAP June 2000 Simulator (Default)
File Edit View Browser Settings Help
Bookmarks Go
Current Session Push WIM WTLS Bookmarks
Clear Session Info
Flush Cache
Content Requests & Responses
Ready

Details of Browser Requests, Responses from servers, and the Local Cache

```
Request: file:/C:/works/mobile/Hello.wml
    Location = [file:/C:/works/mobile/Hello.wml]
    Request Method = [GET]
  Headers (4)
      Accept-Charset = [US-ASCII, ISO-8859-1, ISO-8859-2, ISO-8859-3, ISO-8859-4, ISO-885
      Accept = [application/vnd.wap.wmlc, application/vnd.wap.wmlscriptc, text/x-vCard, text/x-vc
      User-Agent = [Nokia-MIT-Browser/3.0]
      Accept-Language = [en]
Response: 214 bytes, text/vnd.wap.wml; Cp1252
    Location = [file:/C:/works/mobile/Hello.wml]
  Content Data: 214 bytes, text/vnd.wap.wml; Cp1252
```

In this tab, you see details of request and response objects. Using the drop-down list on the right-hand corner you can filter the view on what you want to focus on at that time. When you click **Flush Cache**, you clear the cache of the simulator, ensuring that the new content will be loaded directly from the source rather than cache.

Compilation is normally done by WAP Gateways in real-life scenarios, but the simulators may also provide built-in compilation facilities. When the micro-browser in the device requests the page, the compiled code is sent to it. The micro-browser will decompile this code to render correct WML on the device. Because smaller binary codes are traveling from the WAP Gateway to the device, the pages will be rendered quickly.

Developing & Deploying…

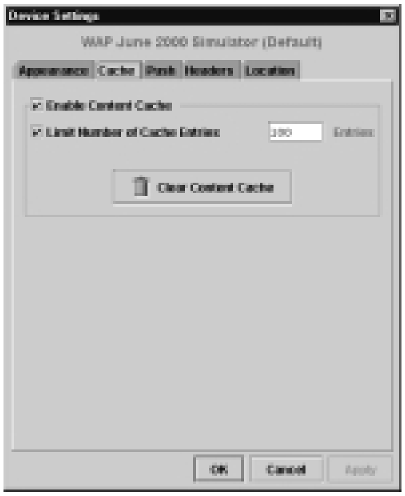
WML Compilation

As discussed earlier, bandwidth is a very premium commodity in a mobile device network. To optimize bandwidth, you need to keep the data transfer sizes to a minimum; thus, WAP content is normally sent to the micro-browsers in compact binary format. The process of converting WML to this binary code (or byte code) is called a *compilation*. Compilation involves a process called *tokenization*, where names of tags and elements are converted to predefined single-character codes. In addition to this, the process of compilation also trims out comments and extra white spaces from the code, and thus, the size of binary file is significantly smaller than the source file.

Configuring Nokia Mobile Internet Toolkit

You can configure each of devices that you have installed with the toolkit to have more control over its individual attributes. For configuring settings, you must first select a device as the active device; you can do this by selecting **Settings | Select Device** and clicking the desired device from the menu. Then, you can select **Settings | Device Settings** to display the configuration sheet for that device. Figure 4.10 displays how you can control different cache attributes for the WAP June 2000 Simulator. You might have different options available for other devices.

Figure 4.10 Device Settings



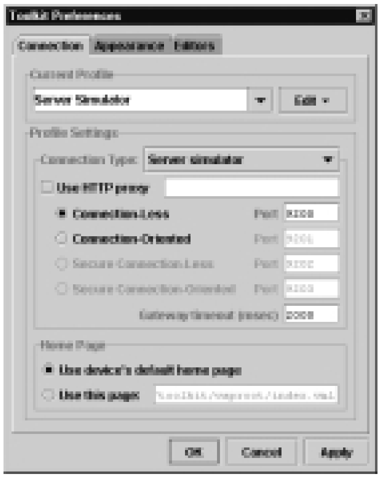
WAP June 2000 Simulator (Default)

✓ Enable Content Cache

✓ Limit Number of Cache Entries 100 Entries

You can configure several attributes of the toolkit by selecting **Settings | Toolkit Preferences** (see Figure 4.11). The most useful tab in this figure is the Connection tab. You can use it to create, edit, and delete connection profiles, which specifies connection type (HTTP, Gateway, or Server simulator and related settings) and home page. You can create several profiles for each of your testing scenarios, for example, you may like to have two profiles where you would use one for connecting to your local Server Simulator and use the other to connect to a real WAP Gateway. This way you can just switch between connection profiles to test your application in different environments.

Figure 4.11 Toolkit Preferences



Toolkit Preferences

Connection | Appearance | Editors

Current Profile: Server Simulator — Edit

Profile Settings

Connection Type: Server simulator

☐ Use HTTP proxy

● Connection-Less — Port 9200
○ Connection-Oriented — Port 9201
○ Secure Connection-Less — Port 9202
○ Secure Connection-Oriented — Port 9203

Gateway timeout (msec) 2000

Home Page

● Use device's default home page
○ Use this page: [illegible]

OK | Cancel | Apply

You can use the Appearance tab to set the look and feel of the main toolkit window. Use the Editors tab to set preferences of the toolkit editor.

Ericsson WapIDE

The Ericsson WapIDE is a set of tools to create and test WAP applications. The IDE includes both a development environment as well as a browser that simulates several Ericsson handsets. The current version of Ericsson WapIDE comes with the support of the following devices:

- R320s
- R380s
- R520m
- T39m

Installing Ericsson WapIDE

To download WapIDE, point your browser to www.ericsson.com/mobilityworld. Browse to Open Zone and choose **Tools and Enablers**; from there, download the file for WapIDE 3.1.1 SDK (WapIDE_311.zip). You will have to register before you can download this file.

System Requirements

To run Ericsson WAP IDE, your system needs to meet the following requirements:

- At least a Pentium II, 266 MHz with 128MB of RAM and 20MB free disk space
- Windows 98, Windows NT, Windows 2000 or higher
- Java 2 Platform, version 1.3.0 or later, which you can download from www.javasoft.com
- Microsoft Internet Explorer 5 or higher for local WML file access.

Running Ericsson WapIDE

Once you have downloaded the file on your computer, you can install WapIDE by running the installer program inside the Zip file. The installation is standard; just follow the on-screen instructions. Once you have successfully installed the Ericsson WAP IDE, you can run the IDE and the browser individually from by selecting **Start | Programs | Ericsson WAP IDE 3.1.1**. Figure 4.12 shows what the initial display of browser looks like.

Using Ericsson WapIDE

The Ericsson WapIDE provides you full-featured development environment for developing your WAP application. It comes with a built-in WAP-enabled editor and also provides debugging support. Once you open the Ericsson WapIDE Application designer from the Start menu, you can create a new project by selecting **File | New | New Project**. When you have written your application, you can validate the syntax by selecting **Tools | Validate Syntax**, and execute the program to see results on the emulator by using **Tools | Test** menu options. Figure 4.13 shows how the development environment looks like with Ericsson WapIDE Application designer and browser.

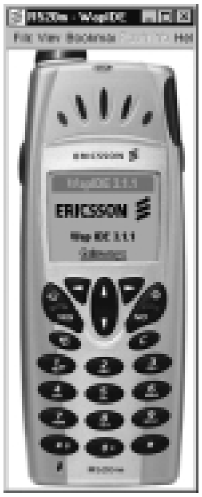


Figure 4.12 Ericsson WapIDE Browser

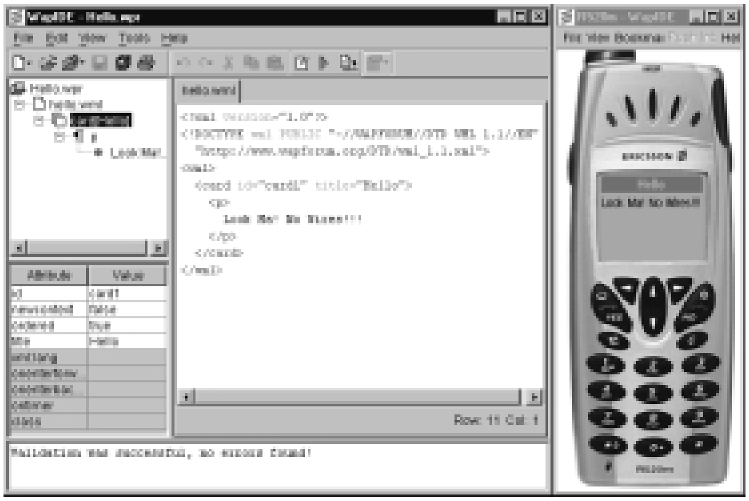


Figure 4.13 Ericsson WapIDE Application Development Environment

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
  "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <card id="card1" title="Hello">
    <p>
      Look Ma! No Wires!!!
    </p>
  </card>
</wml>
```

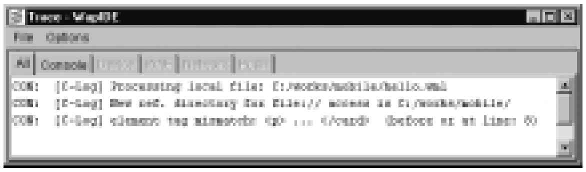
Attribute	Value
id	card1
newcontext	false
ordered	true
title	Hello
xml:lang	
onenterforw	
onenterbac	
ontimer	
class	

```
Validation was successful, no errors found!
```

The WapIDE has several useful tools that you may use while developing applications. The editor window displays the structure of the WML document. The window below it displays attributes and their current values. The bottom window is the message window, where you will be shown messages as the result of commands you selected from the menu.

A trace window will pop up when you run your program—this will provide you debugging information. For example, if you missed the closing tag in the code sample written in Figure 4.13 and tried to run the application, you would see the trace window shown in Figure 4.14.

Figure 4.14 Trace Window Shows Debug Information



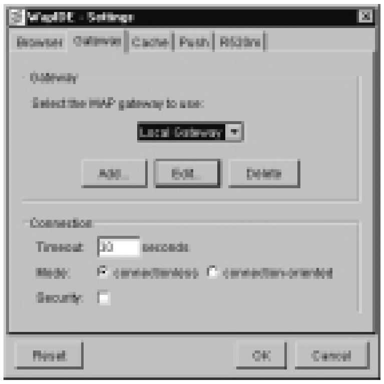
```
COM:  [C-Log] Processing local file: C:/works/mobile/hello.wml
COM:  [C-Log] New cwd. directory for file:// access is C:/works/mobile/
COM:  [C-Log] <element tag mismatch: <p> ... </card>  (before or at line: 6)
```

Configuring Ericsson WapIDE

You can configure each of the WapIDE components using their own menu options. If you need to configure the Application designer, you can do so by launching Application Designer from the Start menu and selecting **View | Settings**, where you will find options for changing the behavior of the editor.

On the other hand, if you wish to configure the simulator, you can do so by launching the browser from the Start menu and selecting **View | Settings**. Here you can set various options through tabs, as shown in Figure 4.15.

Figure 4.15 Configuring Ericsson WapIDE Browser



Select the WAP gateway to use: Local Gateway

Timeout: 30 seconds

Mode: [✓ connectionless] connection-oriented

Security: ☐

For more information on the Ericsson WapIDE, visit the following Web sites:

- The Ericsson Web site at www.Ericsson.com
- Ericsson Mobility World at www.Ericsson.com/mobilityworld
- Ericsson Mobile Internet at wap.sonyericssonmobile.com

Microsoft Mobile Explorer

Microsoft Mobile Explorer (MME) is the Microsoft browser product for the handheld market. For the most part, MME provides a feature set that is similar to that offered by most of the other browsers discussed in this chapter. It does, however, offer one very significant advantage—it is a multimode browser, supporting not only WAP, but also HTML and cHTML. For full details on the capabilities of MME see www.microsoft.com/mobile/phones/mme/.

MME, because of its ease of use, simple configuration, and integration with Visual Studio.NET, is a very productive development environment when you are developing applications using the Microsoft .NET Framework.

Installing Microsoft Mobile Explorer

Microsoft Mobile Emulator is available at www.microsoft.com/mobile/developer/downloads/default.asp as a freely available download. At the time of this writing, its latest version is version 3.0. This download offers two versions of the emulator: a standalone version and a version that is integrated into the Visual Studio.NET IDE.

System Requirements

To run the current version of the Microsoft Mobile Emulator, your system must meet the following requirements:

- Either Microsoft Windows 2000 (SP2 or higher) or, Microsoft Windows XP (RC1 or higher)
- MSXML 3.0 SP1 for the standalone version or Visual Studio.NET Beta 2 for the integrated version
- 10MB of available disk space

Installing the MME Emulator

The Microsoft Mobile Emulator download consists of a single executable file, which runs a standard installation wizard. It includes an option to install either the standalone version, the integrated version, or both. We will use the default option of both. A shortcut to the Microsoft Mobile Emulator will be inserted onto your Start menu and a new submenu will be added to the View menu in Visual Studio.NET.

NOTE

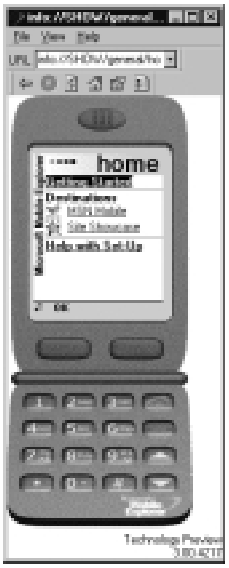
If you do not have Visual Studio.NET installed, you will not be able to choose to install the integrated version of the MME Emulator.

Using the MME Emulator

Start the MME Emulator from the shortcut on the Start menu. This will open two windows: the main browser window containing the mobile phone emulator and the output window that will display various messages and debug information.

When you first start the browser, you will be presented with the MME Info Page, which is accessed from the browser's local file system, as indicated by the info:// prefix in the URL (see Figure 4.16). The first thing you should do is browse some of the links off the home page to get a feel for the browser and the sort of information that is displayed in the output window. You can follow links by either clicking the link or by selecting the link using the up/down arrows and clicking **OK**. Always remember that although you can test your application in the emulator by using a mouse, the end user will in most cases be using a keypad.

Figure 4.16 MME Emulator



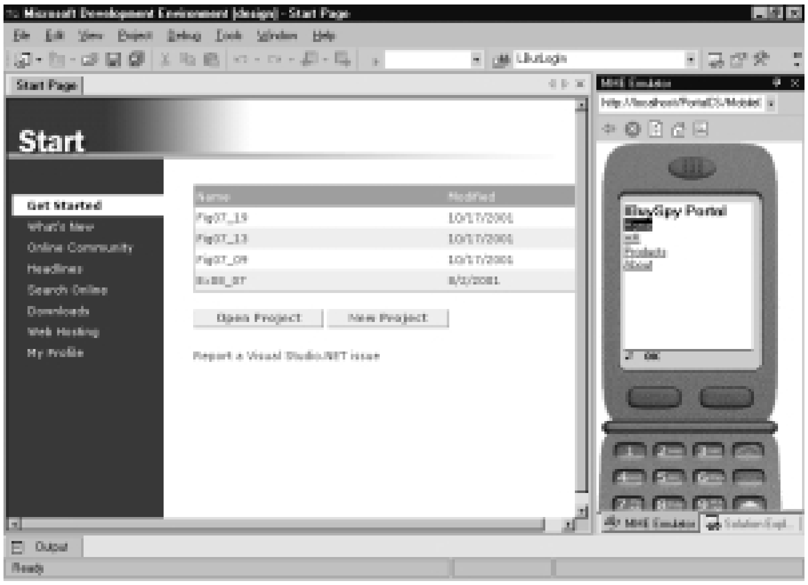
You can change the browser device type by going to **View | Devices** and choosing a browser type. The available browser types are described in the MME Emulator help documentation, which you can access by selecting **Help | Help Topics**.

Running the MME Emulator From Visual Studio.NET

To run the MME Emulator from within Visual Studio.NET, select **View | Mobile Explorer Browser | Show Browser**. This will open the MME Emulator window; this window will behave like any other Visual Studio.NET IDE window, that is, you can dock it inside the IDE.

The behavior of the browser is identical to the standalone version, except that menu options are found under **View | Mobile Explorer Browser**. You can enable and disable the output window (MME Log) by either using the button on the toolbar in the browser window, or by selecting **View | Mobile Explorer Browser | Show Log**. Once again, you can dock the MME Log window with the IDE. Figure 4.17 shows how MME integrates with Visual Studio.NET.

Figure 4.17 MME Emulator Integrates Seamlessly with Visual Studio.NET



Setting the MME Emulator as the Default Project Browser

In order to take advantage of the MME Emulator when a project is run or debugged in the IDE, you need to register MME as a browser in Visual Studio.NET, which you can do by performing the following steps:

1. Select **File | Browse With**.
2. Select **MME**.

If MME is not listed in the browser list, you need to add it as follows:

1. Click **Add** in the Browse With dialog.
2. Browse to the **MmeExec.exe** executable; this is usually in C:\Program Files\Microsoft Mobile Explorer\3.0\Visual Studio.Net\MmeExec.exe.
3. Enter a friendly name that you can recognize later (such as Microsoft Mobile Explorer) and click **OK**.
4. From the browser list, choose Microsoft Mobile Explorer, click **Set as Default**, and then click **OK** to save your changes.

NOTE

You can also register the standalone MME version if you prefer. Simply browse to C:\Program Files\Microsoft Mobile Explorer\3.0\Emulator\mmeemu.exe.

Next, if the project you are working with is a Web or Mobile Application project, you will need to change a debug property to allow browsers other than IE to be used for debugging:

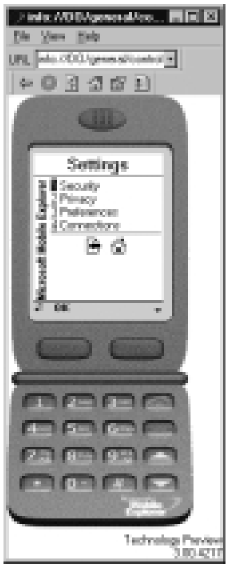
1. Select the Web Application and choose **Project | Properties**.
2. Choose the **Configuration Properties** folder from the tree on the left-hand side of the dialog, and then choose **Debugging** from under this folder.
3. Set the **Always use Internet Explorer** property to **False** and click **OK** to close the dialog.

To test your setup, select **Debug | Start** or press **F5**. The IDE should compile and run the application in debug mode inside the MME Emulator. To stop debugging, you need to select **Debug | Stop Debugging** or press **Shift+F5**.

Configuring MME Emulator

If you need to configure MME Emulator, you can do so by selecting **View | Go To | Settings**. This will open up different options in the emulator screen itself (see Figure 4.18), where you can select and set different settings by using the emulator navigation techniques.

Figure 4.18 Configuring MME Emulator



Microsoft Embedded Visual Tools

The Microsoft Embedded Visual Tools 3.0 delivers a complete desktop development environment for creating applications and system components for Windows CE–powered devices, including the Pocket PC, Handheld PC and Palm-size PC.

The Embedded Visual Tools include embedded Visual Basic and embedded Visual C++, including SDKs for the Pocket PC, Palm-size PC, and Handheld PC. The Embedded Visual Tools are the successor to the separate Windows CE Toolkits for VC++ and VB. This version is standalone and does not require Visual Studio. The Embedded Visual Tools include the Pocket PC, Palm-size PC, and Handheld PC Software Development Kits.

Installing Microsoft Embedded Visual Tools

You can download the Microsoft Embedded Visual Tools from www.microsoft.com/mobile/downloads/emvt30.asp. When you choose to download, you will be prompted for a Microsoft Passport sign-in. When you sign in, you need to complete a small registration form and will then be taken to the download link. Once you download the .exe file, the installation is standard.

System Requirements

The following are the system requirements for Microsoft Embedded Visual Tools:

- A Pentium 150 MHz or higher processor (recommended).
- 24MB memory for Windows 98 Second Edition (48MB recommended). 32MB for Windows NT Workstation 4.0 or Windows 2000 (48MB recommended).
- Adequate hard disk space: **Minimum installation** (embedded Visual Basic and one SDK): 360MB and **Full installation** (embedded Visual Basic, embedded Visual C++, and three SDKs): 720MB.
- Microsoft Windows 2000 Professional; Microsoft Windows NT Workstation 4.0 with SP5, Internet Explorer 5.01, and MDAC 2.1. You can install the Embedded Visual Tools on Windows 98, and you can build your application from there. However, emulation does not work on Windows 98; instead use Windows 2000 or Windows NT as your host machine.

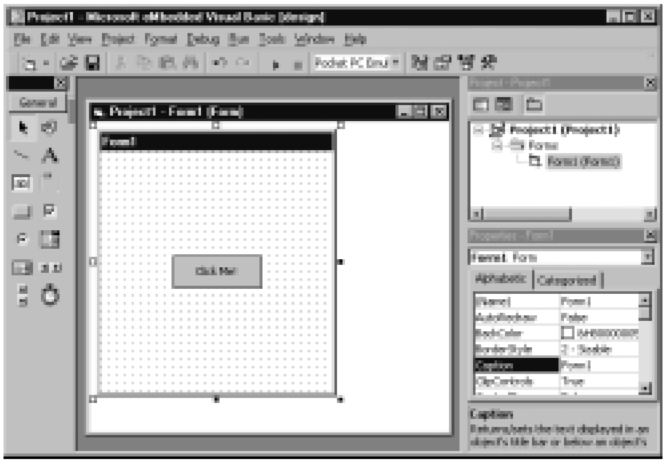
The Pocket PC SDK does not come with support for Jscript installed. If you want to enable Jscript support, you will need to do take few additional steps after the setup of the Pocket PC SDK is complete:

1. Search your computer for a file named Jscript.dll. (If you cannot find it, install the latest scripting release from Microsoft's Scripting Web site at http://msdn.microsoft.com/scripting.)
2. Place the Jscript.dll file in the following directory: C:\Windows CE Tools\wce300\MS Pocket PC\emulation\palm300\windows.
3. Go to the command prompt and navigate to the following directory: C:\Windows CE Tools\wce300\MS Pocket PC\emulation\palm300\windows.

4. Once you are in the directory, you can run the **regsvrce.exe** file found in this directory and passing it the jscript.dll file, **regsvrce jscript.dll**. This will install the patch for Jscript.

Once you have successfully installed the software, you can launch the Embedded Visual Basic development environment by selecting **Start | Program | Microsoft Embedded Visual Tools | Embedded Visual Basic 3.0**. This will open up the development environment, which will look very familiar to Visual Studio Programmers (see Figure 4.19).

Figure 4.19 Microsoft Embedded Visual Basic Development Environment



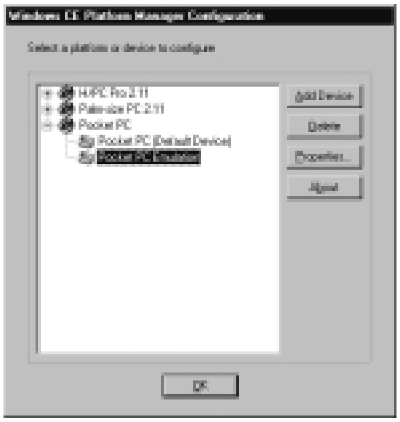
Using the Microsoft Embedded Visual Tools

Embedded Visual Tools provides an easy to use environment for Windows CE–based devices. We give you a simple example to show you the basic development features of this environment. In the Embedded Visual Basic Development Environment, select **File | New Project | Windows CE for Pocket PC Project**. This will open up the Visual Studio–like development environment shown in Figure 4.19. Although you will find most of environment very similar to Visual Studio, a few things are specific to this environment.

Configuring Development Environment

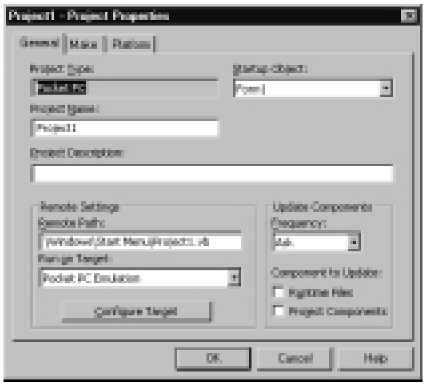
First, you need to configure the environment so that you can test your programs on the specific device, you will be using Pocket PC Emulator device for this example. To do this, go to **Tools | Remote Tools | Configure Platform Manager** (see Figure 4.20), navigate to **Pocket PC | Pocket PC Emulation**, click **Properties**, and then click **Test**. A successful test will pop up the Pocket PC Emulation window and make sure that the connection between development environment and emulator is alive.

Figure 4.20 Windows CE Platform Manager Configuration


Windows CE Platform Manager Configuration
Select a platform or device to configure
H/PC Pro 2.11
Palm-size PC 2.11
Pocket PC
Pocket PC (Default Device)
Pocket PC Emulation
Add Device
Delete
Properties...
About
OK

Next, go to Project Explorer and right-click on project name to access its properties. In the Remote Settings panel, make sure that Remote Path is set to **\Windows\Start Menu\Project1.vb**. You can also select the target device and configure it from there. Click **OK** to confirm the changes (see Figure 4.21).

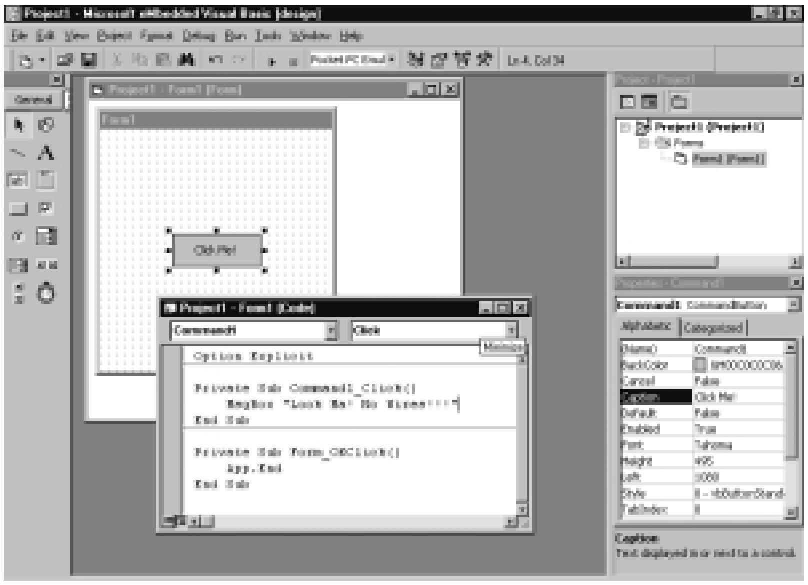
Figure 4.21 Microsoft Embedded Visual Basic Project Properties



Project Type: Pocket PC
Startup Object: Form1
Project Name: Project1
Project Description:
Remote Settings
Remote Path: \windows\Start Menu\Project1.vb
Run on Target: Pocket PC Emulation
Update Components
Frequency: Ask
Component to Update:
☐ Runtime Files
☐ Project Components

Now you are ready to start developing. As a first step, create a command button on the form and change its caption to **Click Me!** using the Properties window. Now double-click the command button to open up the code window and write a command for displaying a message box, as shown in Figure 4.22.

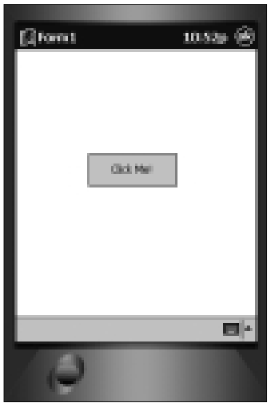
Figure 4.22 A Simple Pocket PC Program



```
Option Explicit

Private Sub Command1_Click()
    MsgBox "Look Ma' No Wires!!!!"
End Sub

Private Sub Form_OKClick()
    App.End
End Sub
```

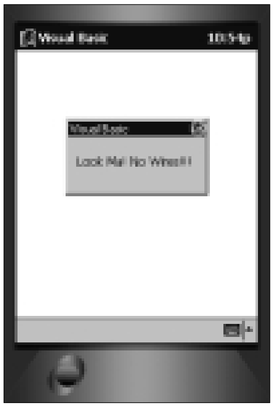
Once you have done this, you are ready to run your program, just press **F5** or select **Run | Start Debug**, and you will see the results on the Pocket PC Emulation device, as shown in Figure 4.23.

Figure 4.23 Program Execution under Pocket PC Emulation



When you click the button, you will see the message box that you included in your program (see Figure 4.24). You can click **OK** to return to the development environment.

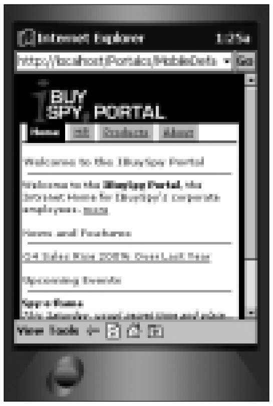
Figure 4.24 Program Execution under Pocket PC Emulation



Using Microsoft Embedded Visual Tools, you can also test your applications directly on a Windows CE device. You can also create a deployment package for a Windows CE device by using the built-in Application Install Wizard**.** You will find these options and more in the Tools menu.

Though the Embedded tools development environment helps you in writing programs, testing, and debugging, you don't need to use the development environment to view your Web pages in Pocket PC Emulator. If you write your programs externally, you can just launch the Emulator by selecting **Start | Programs | Microsoft Windows Platform SDK For Pocket PC | Desktop Pocket PC Emulation.** This will open up the Pocket PC emulator; you can then select **Start | Internet Explorer** from within the emulator to start the browser on Pocket PC. You can use this browser just like you use the desktop browser for browsing your applications. Figure 4.25 shows what the local installation of the IBuySpy sample application (www.ibuyspy.com) looks like in Pocket PC Emulator.

Figure 4.25 Executing External Applications on Pocket PC Emulator



You can keep yourself updated with the latest developments in the embedded tools arena by frequently visiting the following Web sites:

- www.deVBuzz.com
- The Microsoft VBCE newsgroup at microsoft.public.vb.vbce
- The Windows Embedded Community Web site at www.microsoft.com/windows/embedded/community/default.asp

Designing & Planning…

Windows CE .NET

At the time of this writing, Microsoft has announced its Windows CE.NET (also known as code name "Talisker") Beta 2 Technology Preview Program. Microsoft says that it is the platform for developing applications for next generation of "Smart Mobile Devices." Among the new enhancements is support for wireless technologies such as Bluetooth, a feature of device emulation that enables you to emulate the complete device environment without any additional hardware investment, and a Platform Wizard that allows you to select from a number of preconfigured device designs.

For developers, Microsoft has their next version of Windows CE development tools that they call Smart Device Extensions for Microsoft Visual Studio.NET. It offers a scaled-down version of .NET Framework called Microsoft .NET Compact Framework. This brings the power of .NET Framework and languages such as Visual Basic.NET and Visual C#.NET accessible to CE developers. You can also use the powerful development environment of Visual Studio.NET for developing your applications. Building XML Web services and applications on CE is now easier than ever before. However, note that the application written in Embedded Visual Basic 3.0 will not work with this new version. If you wish to port your application, you have to either rewrite them using Visual Basic.NET or wait for an upgrade tool to be released by Microsoft.

A very interesting and important shift is going on for Windows CE developers—you can experience more of it at the following Web sites:

- The Windows Embedded Home Page at www.microsoft.com/windows/Embedded/default.asp
- The Microsoft Developer's Network at msdn.microsoft.com

Summary

Developers use mobile device emulators instead of real mobile devices for most of their development and testing requirements. This is done not only for economic reasons, but also because it increases a developer's productivity by providing a rich environment with various tools available for development and debugging. Emulators also give the opportunity to test an application on a wide variety of devices and enhance/improve application for a particular device before it is too late in the development cycle. This chapter gives you a detailed overview of how to install and use various mobile device emulators—we discussed the following products in this chapter:

- Openwave SDK WAP Edition
- Nokia Mobile Internet Toolkit
- Ericsson WapIDE
- MME
- Microsoft Embedded Visual Tools

This list is by no means complete, and in your development, you may use a few other emulators also. Once you have read this chapter, you should be able to use and develop applications on any of the similar emulators available from other manufacturers.

Solutions Fast Track

Openwave SDK WAP Edition

☑ Openwave SDK WAP Edition is one of the most popular emulators in the market; you can download it from http://developer.openwave.com.

☑ The latest version of Openwave SDK WAP Edition (version 5.0) comes with a sophisticated development environment along with the emulator software.

Nokia Mobile Internet Toolkit

☑ You can download Nokia Mobile Internet Toolkit from http://forum.nokia.com.

☑ Nokia Mobile Internet Toolkit provides an integrated development environment for developing, testing, and debugging applications for Nokia Handset.

Ericsson WapIDE

☑ Ericsson WapIDE includes both a development environment as well as a browser that simulates several Ericsson handsets. The current version of Ericsson WapIDE comes with the support of following devices: R320s, R380s, R520m, and T39m.

☑ You can download Ericsson WapIDE from www.ericsson.com/mobilityworld.

Microsoft Mobile Explorer (MME)

☑ MME is the right emulator for developing and testing all applications targeting MME-based devices.

☑ MME is the only emulator software available that integrates seamlessly with the Visual Studio.NET development environment.

☑ You can download MME from www.microsoft.com/mobile/phones/mme/mmemulator.asp.

Microsoft Embedded Visual Tools

☑ Microsoft Embedded Visual Tools lets you develop applications for Microsoft Windows CE–powered devices such as Pocket PC, Handheld PC, and Palm-size PC.

☑ Embedded Visual Tools comes with a Microsoft Visual Studio–like development environment and provides Embedded Visual Basic and Embedded Visual C++ as the programming languages for development.

☑ You can download Microsoft Embedded Visual Tools from www.microsoft.com/mobile/downloads/emvt30.asp.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: Which emulator should I use?

A: There is no single *right* emulator. You will have to use several emulators and also real devices to make sure that your application is performing its desired behavior in all of them. While you are developing, you can stick to one emulator that you feel most comfortable working with, but during the later stages of development and testing, we recommend that you get a look-and-feel of your application in as many environment as possible.

Q: Where can I find more information about emulators?

A: The most definitive source for information on an emulator will be its manufacturer. Companies keep on releasing new versions to meet new standards and requirements. They normally maintain developer forums and discussion groups on their Web sites to help developers in keeping up with latest technologies. Bookmark and regularly visit the Web sites of popular device vendors. Here are some other Web sites you might find useful in your development work:

- www.googlegroups.com
- www.wapforum.org
- www.wirelessdevnet.com

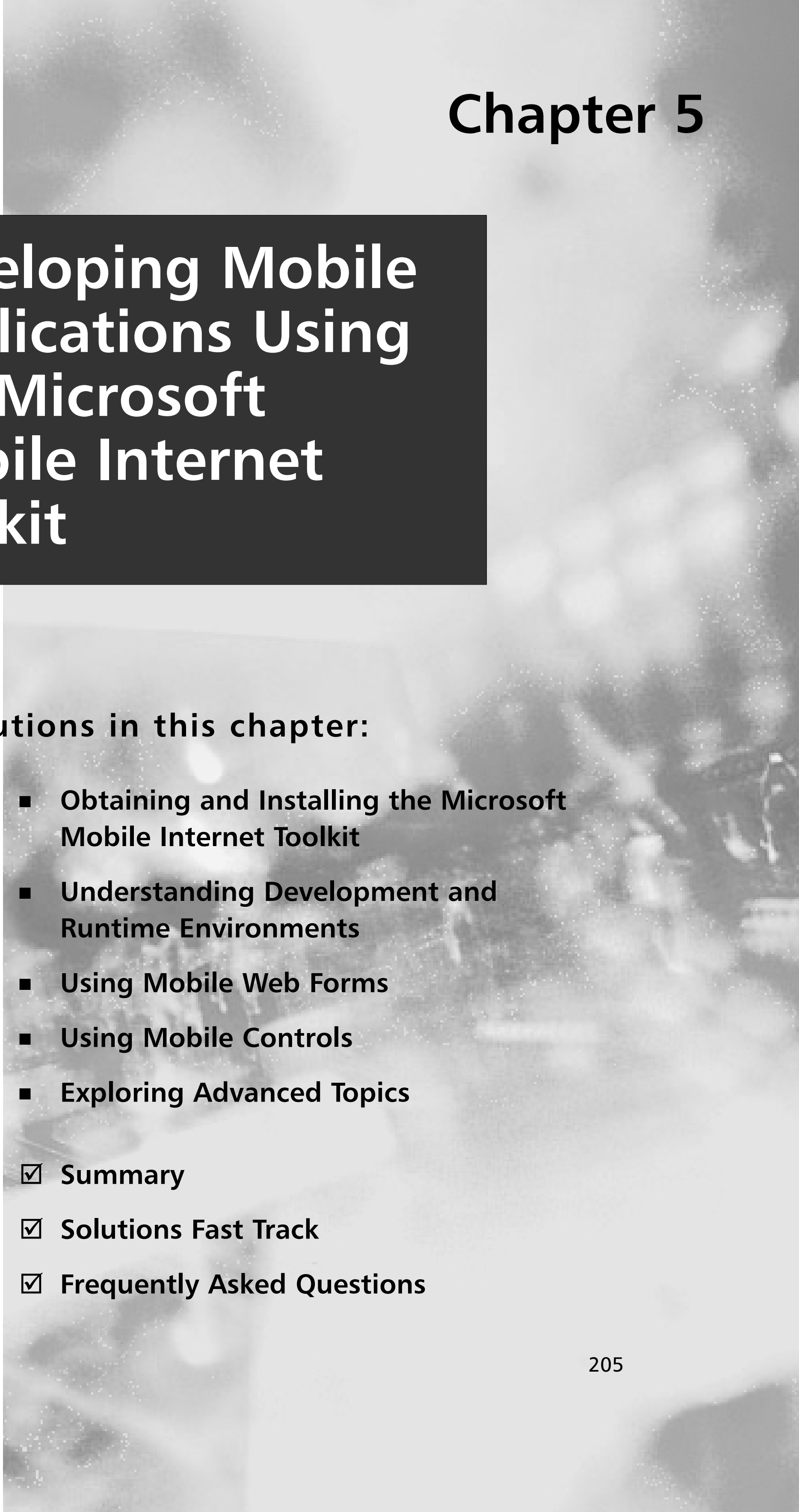
Q: One of the download locations given in this chapter does not work. Where can I find the correct one?

A: The download locations were available and working at the time this chapter was written, but due to the changing nature of the Web, some links might be moved to a different location. In that case, you can check several places:

- Check for updates on this book's accompanying Web site.
- Go to the manufacturer's home page and try to search for the specific product through the search option.
- Try searching for the product by typing its name in a search engine such as www.google.com.

Chapter 5

Developing Mobile Applications Using the Microsoft Mobile Internet Toolkit



Introduction

Microsoft's Mobile Internet Toolkit provides an environment for the rapid delivery mobile Internet applications to multiple device types, through a single application model. In Mobile Internet Toolkit version 1.0, support for device markup languages include WML (Wireless Markup Language), HDML (Handheld Device Markup Language—only through gateway transcoding), cHTML (Compact HTML—for IMODE devices) and regular HTML Web browsers. In the future support for XHTML (Extensible HTML—WAP 2.0) and other markup languages will also be possible both from Microsoft software enhancements and the extensibility features of the Mobile Internet Toolkit.

The Mobile Internet Toolkit and its components are built on top of .NET Framework and ASP.NET, and attempt to address the many challenges and considerations discussed in the preceding paragraph with developing and delivering Mobile Internet applications. These challenges and considerations include the various wireless network carriers, protocols types, and device capabilities. Delivering applications that work consistently with the various device types can be an overwhelming challenge, especially in North America with traditional techniques of generating the various mobile Internet markup languages.

The Mobile Internet Toolkit is specifically for delivering real-time browser based applications. This means that there must be a wireless Internet connection for application features to run. If you are out of wireless network coverage, the application cannot be used. This is a significant disadvantage, but with improving wireless network coverage and increasing availability of mobile Internet-enabled devices, the Mobile Internet Toolkit provides significant value. For more complex/offline mobile applications, you can use different technologies. Microsoft is developing the .NET Compact Framework, an embedded application environment for mobile devices. The Compact Framework will allow any .NET Win Form application with the appropriate form factor to run locally on any device with the .NET Common Language Runtime (CLR). This will allow you to create an embedded mobile application that will work on multiple mobile devices, especially current Pocket PC 2002 devices (with appropriate CLR install) and CE4.0 devices due out in 2002. Compact Framework applications contain the ability to run without any wireless network connection and store data locally. This makes it more suitable for applications for complex mobile sales forces or field forces where coverage can be an issue. The Compact Framework is not covered in this chapter; the focus is on browser based applications and the Mobile Internet Toolkit.

The Mobile Internet Toolkit platform has development and runtime environments. The development environment contains Mobile Web Forms and Mobile Controls allowing for simplified development of mobile Internet applications. The Mobile Designer, which comes as part of the Mobile Internet Toolkit, and Visual Studio.NET provides a visual integrated development environment (IDE) called the Mobile Designer for the rapid implementation of mobile Internet applications. The runtime environment provides device identification and content rendering capabilities to deliver mobile Internet applications to multiple device types without consideration during application development.

The key features of the Mobile Internet Toolkit platform are as follows:

- **Single application model** Mobile Controls and Visual Studio.NET allows for the creation of single-source application code that works with multiple device types by generating various markup languages including WML, cHTML, and standard HTML.
- **Rich device identification** The Mobile Internet Toolkit runtime detects the connecting device markup language and devices features (such as support for telephony features). This device detection allows the Mobile Internet Toolkit to dynamically customize content.
- **Server-side technology** Mobile Internet Toolkit applications run server-side, markup languages are dynamically generated on the server and delivered to various mobile Internet devices.
- **Enhanced services to meet mobile needs** Support for WAP (Wireless Application Protocol) is a key feature of the Mobile Internet Toolkit; along with WAP, other enhanced services optimize the delivery of mobile Internet applications.
- **Push interface to and integration with mobile information server** Applications requiring notification functionality can interoperate with Mobile Information Server for secure alert delivery. This interface can be through COM or SOAP and allows a Mobile Internet Toolkit application to have integrated data pull and push functionality.
- **Extends Visual Studio.NET to mobile devices** Mobile Internet Toolkit is build on top of ASP.NET and makes use of all development, debugging, and testing features of VisualStudio.NET. You can leverage all the programming languages and libraries available in Visual Studio.NET with the Mobile Internet Toolkit.

- **Mobile controls and designer** Mobile Controls, as with any .NET Controls, allow for the rapid development of applications. Using the Mobile Designer, you can quickly add and arrange Mobile Controls within a mobile Internet application.
- **Integration of device emulators** For rapid mobile Internet application development, device emulators such as the Microsoft Mobile Internet Explorer emulator can be integrated with Visual Studio.NET. This allows you to easily test and preview a mobile Internet application during development. Pocket PC emulators are available within the Microsoft Embedded Visual Tools, but expect to see standalone Pocket PC emulators on the market shortly.

This chapter provides a solid introduction and overview of the key features of Microsoft's Mobile Internet Toolkit. We step through the process of installing and configuring the Mobile Internet Toolkit and well as creating Mobile Web Forms, Mobile Controls, and deploying the Mobile Internet Toolkit application. We cover advanced topics and techniques. We also cover best practices, as well as emphasis on important considerations for delivering mobile Internet applications. Why Use the Microsoft Mobile Internet Toolkit? If you are new to mobile application development, or if you have been developing mobile Internet applications with technologies such as WML, the reasons why Microsoft has developed this platform and the value it brings is worth understanding. If you are developing mobile Internet applications directly with WML, supporting multiple device types and platforms can be a challenging situation. Creating a WML mobile Internet application that supports multiple devices types involves creating multiple applications versions; additionally, you will need to embed most application logic with Active Server Pages/XML/XSLT to dynamically generated content. This provides for very complex situations for the delivery of mobile Internet applications, especially consumer services such as a consumer stock trading mobile Internet application.

Microsoft has for a number of years been working with many of these challenges. This began with the XSL ISAPI filter that allowed for partial device independence in a mobile application. The Mobile Internet Toolkit is the evolution and convergence of these ideas to simplify the development and delivery of mobile Internet applications. By taking care of the device detection, the use of various content types and the customization of content depending on the connecting device, the Mobile Internet Toolkit provides an ideal environment for the delivery of mobile Internet applications.

Mobile Internet Toolkit provides the following benefits that are transparent to the developer and organization implementing mobile Internet applications:

- **Support for different carriers and wireless network types** There are many different carriers around the world. In the North American market, most carriers use different wireless network technology (such as GSM, CDMA, TDMA, CDPD, IDEN, Mobitex, DataTAC, GPRS, CDMA2000 1XRTT, and so on). Mobile Internet Toolkit applications will work on top of all of these networks without any Mobile Internet Toolkit application development considerations.
- **Support for different protocols and markup languages** Many protocols and markup languages exist in the mobile Internet world. Mobile Internet Toolkit leverages multiple markup languages WML, HTML and cHTML delivering customized content depending on the connecting device. Again, this is independent of the creation of the mobile Internet application.
- **Support for different micro-browser implementations** Even within the WAP standard are many different ways that vendors have implemented this standard in their micro-browsers. Mobile Internet Toolkit detects these differences in micro-browsers and customized content.
- **Effective development/debugging capabilities** When developing many types of mobile Internet applications using the various technologies available, diagnosing runtime-error and debug-coding problems is often a challenge. For example many mobile Internet-enabled phones will display an error message "A problem occurred, please contact administrator." Mobile Internet Toolkit, when integrated with VisualStudio.NET and the Mobile Designer, takes advantage of all the .NET development/debugging capabilities. This includes using breakpoints, stepping through the code, debugging, rich error messages, and other advanced application features.
- **Support for different mechanisms of session/state management** Support for cookies and sessions with many mobile Internet devices is random at best. This is partly because some carrier WAP gateway's are involved in state management and often have poor support. The Mobile Internet Toolkit contains the ability to use cookieless sessions so that state can be maintained independent of the carrier gateways.

Obtaining and Installing the Microsoft Mobile Internet Toolkit

The implementation of the Mobile Internet Toolkit involves a separate installation on top of the .NET Framework. This is true both for development and production machines, with some additional installs for the development environment when using Visual Studio.NET. You can also use any WAP phone/device emulator available from the various vendors, such as the Mobile Internet Explorer emulator. You can obtain the Mobile Internet Toolkit directly from Microsoft—go to www.microsoft.com/mobile and follow the developer downloads links.

The Mobile Internet Toolkit runs on top of the .NET Framework and ASP.NET and hence requires that you have the .NET Framework on your servers in both development and production environments. If using .NET Server (WinXP), this new operating system version comes with the .NET CLR, and hence for the .NET Server, you need to install only the Mobile Internet Toolkit for production. Windows XP developer machines are identical. However, if you want to use the Mobile Designer and other advanced development features of VisualStudio.NET, you need to install Visual Studio.NET.

If you are planning to stay with a Windows 2000 environment (at least initially), then you need to do a little more to get the Mobile Internet Toolkit environment going. You will need to have Windows 2000/IIS5, .NET Framework, and then the Mobile Internet Toolkit. Again, for development, you can also install Visual Studio.NET.

As with any Microsoft software products, minimum hardware requirements exist for installing, developing, and deploying Mobile Internet Toolkit applications. For more information on hardware requirements, see specific Mobile Internet Toolkit documentation. However, having production MMIT servers that mirror the server hardware you typically would see in a Web application environment is a recommended best practice. Mobile Internet Toolkit applications are almost identical to existing Web applications in hosting, performance, and other application features.

The actually installation process of the Mobile Internet Toolkit version 1.0 is very straightforward once you have the platform installed and running. When you execute the Mobile Internet Toolkit installation file, you will be prompted to accept the licensing agreement and other prompts. The installation will run smoothly and will confirm when completed. The installation of the Mobile Internet Toolkit installs the runtime environment and well as the Mobile

Controls and project features (development environment) when using Visual Studio.NET on the development machine.

Device emulators are very useful for the development and testing of mobile Internet applications. After you have installed the Mobile Internet Toolkit, you can install any number of device emulators on the development machines. Microsoft provides a device emulator called the Microsoft Mobile Explorer (MME) emulator as part of their mobile solutions offering. The MME includes support for HTML and WML. For more information on emulators, see Chapter 4.

Within the Mobile Internet Toolkit, support for the following device types is available in version one. Specific devices can vary, so be sure to check Mobile Internet Toolkit documentation. Supported device types include WAP, IMODE, Pocket PC (including Pocket PC 2002), PALM, RIM Blackberry, emulators, and standard desktop Internet Explorer versions.

Understanding Development and Runtime Environments

The Mobile Internet Toolkit has a development and runtime environment. The separation of these environments within the Mobile Internet Toolkit provides a layer of abstraction between how the application is written and how it is rendered on the target mobile Internet devices. In the development environment, a number of steps are involved in the process of application development: creating a Mobile Web Form, adding mobile controls for application UI functionality, integrating business logic, testing on emulators and target devices, and finally posting to the deployment Web server. As we discussed in the preceding section, this server would be a Windows 2000 server with the .NET Framework or a .NET Server and Mobile Internet Toolkit. On the production side (runtime environment), mobile Internet devices would make an HTTP request to the Mobile Internet Toolkit application. A standard *.aspx file name would be used to reference the Mobile Internet Toolkit application. The runtime environment would detect the client capabilities, execute the mobile application, and then use the appropriate device adapter to return appropriate content type in an HTTP response. Within the production environment you have the ability to update device capabilities and device adapters, in this way, the Mobile Internet Toolkit platform can be easily extended to include new devices and application functionality.

Using Mobile Web Forms

Mobile Web Forms are the foundation of developing a mobile Internet application with the Mobile Internet Toolkit, and they significantly simplify the development of mobile Internet applications. The Mobile Web Form provides a simplified development approach using Mobile Controls.

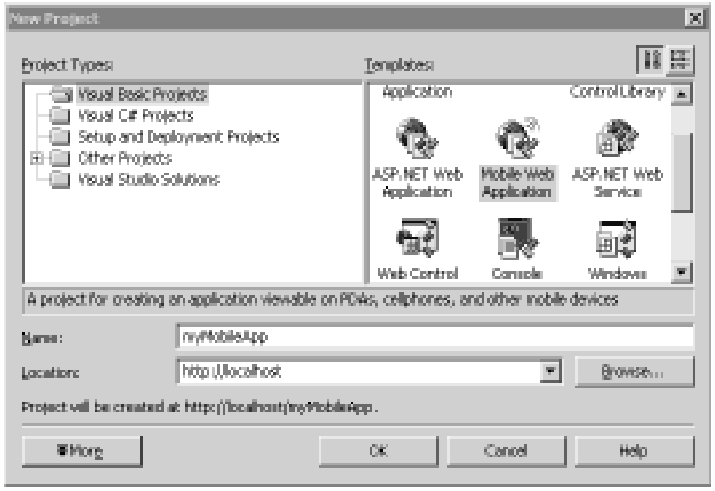
If you are familiar with WML (Wireless Markup Language), many of these runtime controls map directly to WML functionality. The Mobile Web Form relates to a WML deck and the Form control is analogous to a WML card. Within your application, you can combine these simplified Mobile Controls, which reduces the need to learn and know in-depth details of WML and other mobile Internet markup languages; these details are all handled by the Mobile Internet Toolkit. If you are not familiar with WML, the good news is that you don't need to be. Mobile Web Forms and Mobile Controls create a layer of abstraction. You will develop your mobile Internet applications by using Mobile Controls within the Mobile Web Form; the Mobile Internet Toolkit handles the rest by delivering the appropriate content and formatting depending on the target device. A Mobile Web Form is a single ASP.NET file within your Mobile Web application project, it has the extension *.aspx and would be referenced from a mobile Internet device with a URL.

If you are using VisualStudio.NET, the first step in creating a Mobile Web Form is to create a Mobile Web application project. This is done in the same way that you create a regular Web application. However, after you have installed the Mobile Internet Toolkit, you will see a new project type called Mobile Web Application. Select this application type and server location, and the project will be created. See Figure 5.1 for an example of creating of a Mobile Web Application project. The new project will then be created and necessary files added to the project. Within the project explorer window, notice the Web.Config file. In a Mobile Web application, the Web.Config file contains the regular Web configuration information as well as device filters. The device filters are what allows your application to detect device types via appropriate header information and deliver customized content to difference devices.

Once you have created a Mobile Web application project, you can then add Mobile Web Forms to your project. Figure 5.2 shows a Mobile Web Form with multiple form controls. To create a Mobile Web Form, just right-click on project name in the project explorer and select **Add New Item**. You can then select a Mobile Web Form, which is analogous to a WML deck. A Form Control is analogous to a WML card. Mobile Web Forms are mainly just a container for Mobile

Controls, however, a number of other features of a Mobile Web Forms need to be considered.

Figure 5.1 Creating a Mobile Web Application with VisualStudio.NET



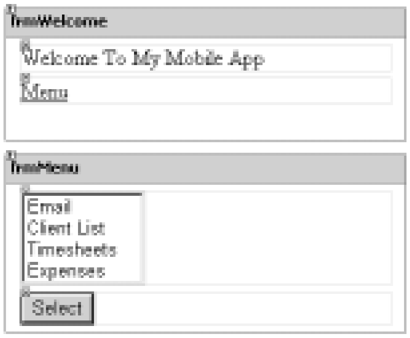
A project for creating an application viewable on PDAs, cellphones, and other mobile devices

Name: myMobileApp

Location: http://localhost

Project will be created at http://localhost/myMobileApp.

Figure 5.2 Mobile Web Form in the Mobile Designer



As with a regular ASP.NET Web Forms, Mobile Web Forms have Register and Page directives. The following code is the default example of the Web Form directives that are created automatically when you create a Mobile Web Form:

```
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
    Assembly="System.Web.Mobile" %>
```

```
<%@ Page Language="vb" AutoEventWireup="false"
    Codebehind="Default.aspx.vb"
    Inherits="myMobileApp.MobileWebForm1" %>
```

Notice a number of points about these Mobile Web Form directives. The tag prefix for mobile controls is "mobile". With the *Namespace*, you can see that the inheritance path of the Mobile Web Form controls is *System.Web.UI.MobileControls*. Within the Page directive, you can see the language and *Codebehind* properties, which define the default language and the code source connected to the execution of the page.

When developing the business logic within a Mobile Web Form, you can use any of the 20+ programming languages that are compatible with the .NET Common Language Runtime. You should use the Code-Behind functionality in Mobile Web Forms; it allows you to separate the presentation logic from the business logic and makes for much easier application development.

Using Mobile Controls

Now that you have created a Mobile Web Application project and a Mobile Web Form, you can begin adding Mobile Controls to define and create application functionality and user interface (UI). As we have touched on a couple of times earlier in this chapter, the Mobile Internet Toolkit simplifies the development of mobile Internet applications. It does this with the use of Mobile Controls.

Mobile Controls create a layer of abstraction between the code that you as a developer would be creating and the markup language and specific device codes sent to connecting devices. As you saw with Mobile Web Forms, many mobile controls are analogous to HTML and WML functionality. Another example of this is the Call Control for WAP devices that support WTAI (Wireless Telephony Application Integration) functionality, the Call Control allows you to easily add support for dialing phone numbers directly from your mobile Internet application. This is especially useful if you are connecting to some kind of Voice Internet application (VoiceXML) directly from your mobile Internet application.

You can develop Mobile Controls with various IDEs, including text editors and other Web development environments. Microsoft includes an IDE for developing Mobile Internet applications in the Mobile Internet Toolkit called the Mobile Designer. The Mobile Designer is installed into Visual Studio.NET when the Mobile Internet Toolkit is installed.

Using the Mobile Designer, you can drag Mobile Controls onto your Mobile Web Form. If you drag an incorrect control for the section of the Mobile Web

Form, you will get an error, which you can easily correct by deleting and dragging the control again to the correct position within your application.

In addition to the drag and drop of Mobile Controls, you can perform direct code manipulation and optimization via the source code editor. Because the Mobile Designer is part of VisualStudio.NET, you have the full debugging capabilities available for WinForms and ASP.NET applications. When compilation and runtime errors occur, rich error messages allow for effective development.

You can manage code as part of the mobile solution in the same mechanism as with any Visual Studio.NET application. If you are just getting started with the VisualStudio.NET environment, VB.NET is often the best way to get started programming. However, if C++ or C# is your preference, you can leverage these languages for developing mobile Internet applications.

The following section outlines all of the controls in the Mobile Internet Toolkit v1.0, which we have organized into basic and advanced controls. We also mention custom controls later in the section and how you can create and use them. We describe the basic features, but for more information, see the Mobile Internet Toolkit documentation.

Understanding Basic Controls

The basic controls provide the basic functionality used in most mobile Internet applications. They include the following:

- **Call** Allows phone numbers to be dialed directly from the application with devices that support calling.
- **Command** As with WinForm and Web Form applications, command controls allow you to program against a user action. You can display a command control as a button or link.
- **Form** Encapsulates other mobile controls and is typically the smallest unit of display in a mobile Internet device. Is analogous to a WML card.
- **Image** Allows for the use of image files in the mobile Internet application. Typically used with device-specific customization; common file types are wireless bitmaps (WBMP).
- **Label** Typically used for headings and display small amounts of information in the device.

- **Link** Allows the user to move to another form control or Mobile Web Form file. However, doesn't contain events seen with the command control for programmable user actions.
- **List** Allows for the display of a list of items that don't require any action.
- **Panel** Is similar to the Form control, it is used to encapsulate other controls and will force controls within the panel to be rendered together.
- **MobilePage** Is the primary control for the entire mobile page, providing the outermost containers for the mobile Web application. The mobile page is specified in the page directives.
- **ObjectList** Is similar to the List control, however, you can link list items and do other functionality to give the user more information.
- **SelectionList** Allows the user to select an item from a list, which can be a drop-down, selection list, or radio button–type functionality but will display differently depending on the target device.
- **TextBox** Used to allow user input that can be posted to the server for processing, such as a username and password.
- **TextView** Used to display more than a couple of lines of text, such as memo fields.

Understanding Advanced Controls

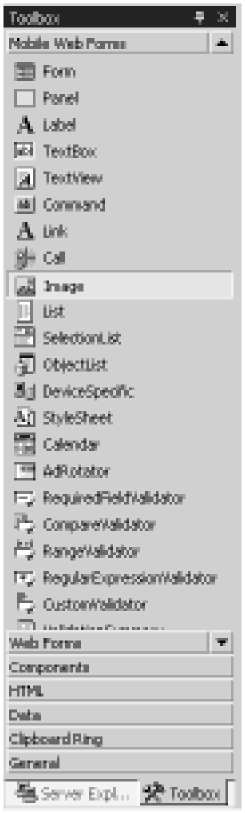
Advanced controls consist of specific controls used for advanced application functionality. They include the following:

- **AdRotator** As with Web Forms, you can use the AdRotator to rotate advertisements on a mobile Internet device. However, with limit Form Factors on my mobile Internet device, advertisements are not that practical.
- **Calendar** Allows for the selection of dates. On a WAP–enabled device, the user is presented with separate cards for Year, Month, and Day selection. On a Pocket PC, a grid allows users to select day direction or navigate to another month or year.

- **StyleSheets** Allows you to specify external stylesheets for use in a Mobile Internet Toolkit allowing for complete customization of the UI.
- **Validation Controls (CompareValidator, CustomValidator, RangeValidator, RegularExpressionValidator, RequiredFieldValidator, ValidationSummary)** All of these validation controls listed allow various types of validation to occur within the Mobile Internet Toolkit application. Typically, they are tied directly to another control such as a *TextBox*, so the validation occurs with input related to the *TextBox* control.

When using the Mobile Designer, you just drag and drop the appropriate controls onto your Mobile Web Form. Figure 5.3 shows the Mobile Web Controls toolbox showing all the built-in controls that come with version 1.0 of the Mobile Internet Toolkit.

Figure 5.3 Mobile Web Controls Toolbox



So, let's start creating a basic mobile Internet application using the Mobile Web Forms and Mobile Controls. Continuing with the Mobile Web Form created in the previous section; add a *Label* and *Link* control to the form. Here is the code that is created by the dragging and dropping of the *Form*, *Label*, and *Link* controls:

```
<mobile:Form id="frmWelcome" runat="server">
          <mobile:Label id="lblWelcome" runat="server">
Welcome To My Mobile App
</mobile:Label>
<mobile:Link id="lnkMenu" runat="server" NavigateURL=
    "#frmMenu">Menu</mobile:Link>
     </mobile:Form>
```

As you can see, this is a welcome screen with a link that leads to a menu of items. See Figure 5.2 or Figure 5.4 to see how these controls look in the VisualStudio.NET Mobile Designer.

The *Label* control is used to display the "Welcome to My Mobile App" message; the *Link* control is used to link the user to the *frmMenu*. This demonstrates typical application functionality for a mobile Internet application. You need to limit the amount of information and action in any Form control. As we have discussed, the Form control is analogous to a WML card, so it is the smallest unit of display in the device screen. After the user selects the Menu link, they move to the *frmMenu*. To create *frmMenu*, you would drag another Form control onto the Default.aspx Mobile Web Form. To the *frmMenu* add a Label, Selection List, and Command control. Here is how the *frmMenu* would look:

```
<mobile:Form id="frmMenu" runat="server">
<mobile:SelectionList id="SelectionList1" runat="server"
SelectType="ListBox">
               <Item Text="Email" Value="1" ></Item>
               <Item Text="Client List" Value="2" ></Item>
               <Item Text="Timesheets" Value="3" ></Item>
               <Item Text="Expenses" Value="4" ></Item>
          </mobile:SelectionList>
<mobile:Command id="cmdSelectMenu" runat=
    "server">Select</mobile:Command>
</mobile:Form>
```

This form would present the user with a list of options (Email, Client List, Timesheets, and Expenses). The user could then select the item and be transferred to the appropriate section. Notice that both of the Form controls (*frmWelcome* and *frmMenu*) are on the same Mobile Web Form, this is different from typical ASP.NET pages, which typically have one Form control per Mobile Web Form. You can have as many Form controls within a Mobile Web Form (*.aspx file) as you would like. However when dynamically generating lists and other content in your Mobile Internet Toolkit application, spreading Form controls and other application functionality over multiple Mobile Web Forms makes more sense. In the case of the preceding example, when the user selects one of the items on the list, they are transferred to a different Mobile Web Form (such as Timesheets.aspx) that would contain multiple controls with the relevant timesheet functionality.

You can build business logic into your Mobile Internet Toolkit application by using any of the programming languages in VisualStudio.NET. The application in this chapter uses VB.NET as the default programming language. Within your Mobile Internet Toolkit application, you could program any number of application features and properties. For example, the *Command* control in the preceding example could involve dynamic functionality by programming the *On_Click* event. The event code could contain any type of business logic but would use the following format:

```
Private Sub cmdSelectMenu_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs)
    Handles cmdSelectMenu.Click


     'Business Logic…


     Response.Redirect = "Timesheets.aspx"
     'ActiveForm = timesheet


End Sub
```

This subroutine could be embedded within <script> tags directly in the Mobile Web Form, but you should use the Code-Behind features for encapsulating business logic in Visual Studio.NET.

As you can see in this code example, when the *cmdSelectMenu* Command control is clicked, this private subroutine is executed. The business logic is executed, and the user would be either redirected to the Timesheets.aspx Mobile

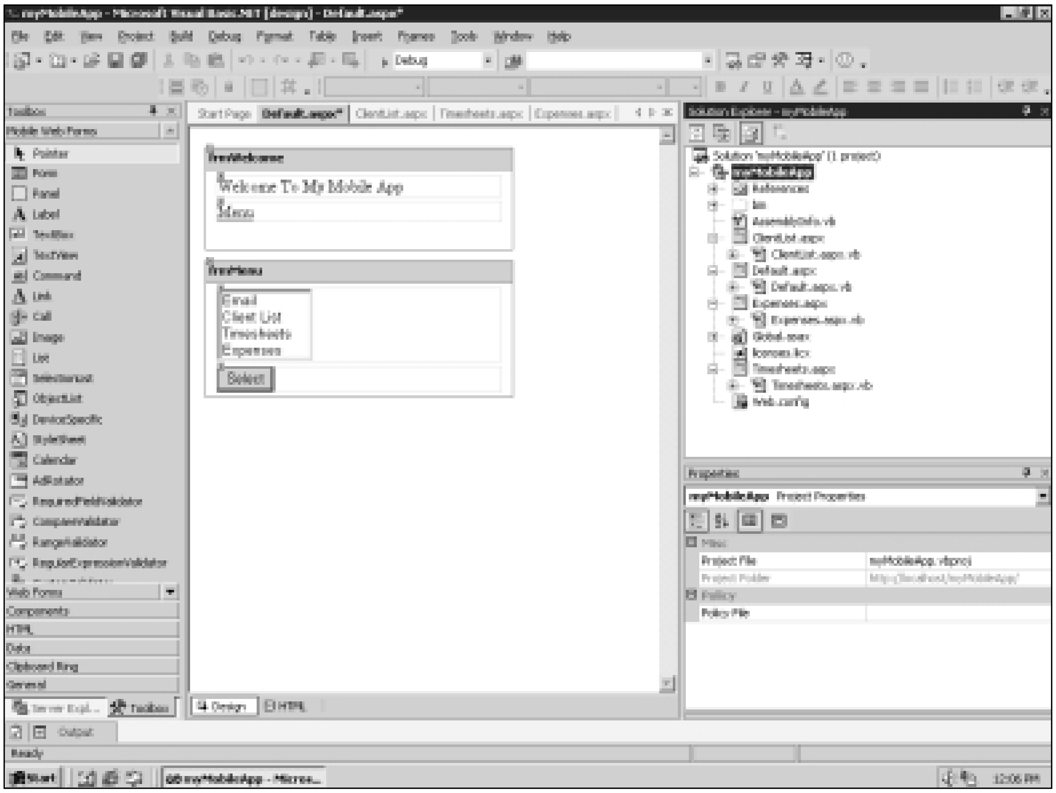
Web Form or to the timesheet Form control on the same Mobile Web Form. This example leads into a short discussion about application navigation.

Within a Mobile Internet Toolkit application, the user navigation is very limited. If using a WAP-enabled cell phone, this normally involves only two possible actions: Select or Options (normally providing a secondary option). Hence, this is the reason why you embed multiple Form controls within a Mobile Web Form. The user would step through multiple Form controls to do a particular action. A good example of this is the Calendar control; when using a WAP-enabled device, the selection of a date would involve a Form for selecting the year, a Form for selecting the month, and another Form control for selecting the day. The Calendar control does this automatically, but within custom application functionality that you may be developing, you need to easily move between Form control and even Mobile Web Form files. As the preceding example demonstrated, to move to another Mobile Web Form you would just use *Response.Redirect.* If you want to pass variables, you can do it on the query string. To move between different Form controls within a Mobile Web Form, you just need to specify *ActiveForm=*. Because you are still within the same Mobile Web Form, all the local variables still apply to the *ActiveForm* control. The *ActiveForm* property is important in the development of a mobile Internet application; by using the *ActiveForm* property, you are able to present the user with relevant data and/or further options.

Before ending the discussion on creating and using Mobile Controls, we need to cover a few other areas. Figure 5.4 shows our Mobile Internet Toolkit application in the Mobile Designer within Visual Studio.NET.

As you can see, the Mobile Web Form is loaded in the Mobile Designer. On the right, you can see the project explorer showing all the files in the Mobile Web Application project. Below that is the Properties window. The Properties window is very useful in this environment for defining the properties of the Mobile Web Forms and Mobile Controls. For example, when you add an *Image* control, you can use the Properties window to specify the image file location, device-specific content, relevant text for devices not supporting images, and so on.

Figure 5.4 The Mobile Designer/Visual Studio.NET Environment



Developing & Deploying…

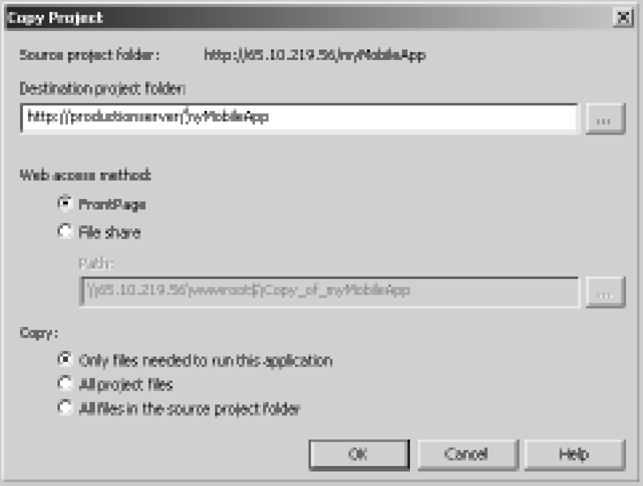
Mobile Internet Toolkit Applications

You can perform mobile Internet application deployment in a number of ways. If you are using Visual Studio.NET and the Mobile Designer, just use the Web Copy wizard in the Project drop-down. In Figure 5.5, you can see this interface. Note the ability to transfer files using server extensions as well as the ability to copy all project files or just the files to run the application. If you just copy the files to run the application, only the compiled runtime files will be copied to the production server. In addition to the deployment using Visual Studio.NET, you can also use XCOPY or any mechanism to copy runtime files; however, using Visual Studio.NET is recommended. After you have deployed the application to

Continued

the appropriate production Web server, you can then access it from your wireless Internet-enabled device using the relevant URL.

Figure 5.5 Visual Studio.NET Project Copy Wizard


Copy Project

Source project folder: http://65.10.219.56/myMobileApp

Destination project folder: http://productionserver/myMobileApp

Web access method:
[✓ FrontPage]
File share

Path: \\65.10.219.56\wwwroot$\Copy_of_myMobileApp

Copy:
[✓ Only files needed to run this application]
All project files
All files in the source project folder

Exploring Advanced Topics

In addition to the basic concepts of the Mobile Internet Toolkit and the Mobile Controls we have covered, a number of more advanced topics are important when developing a mobile Internet application.

Pagination

Pagination is a term used to describe automatic grouping of data depending on device capabilities. You can use the built-in paging properties or define custom pagination. For example, if a WAP phone supports only 10 lines of data display, you can make the application automatically limit data lists to 10 lines. If a list contains more than 10 lines of data, you could create a next/back option so that the user can move through the dataset customized for the device. The Mobile Internet Toolkit will paginate lists and other mobile Internet application features to fit target device form factors.

The *List*, *Objectlist*, and *TextView* controls feature automatic pagination in which content will automatically be paginated by the Mobile Internet Toolkit. In addition to the automatic pagination, you can also set up custom pagination by setting the *ItemCount* property to the appropriate number of items and using the *LoadItem* event you can action a particular dataset.

Device Specific Content and Customization

Within the Mobile Internet Toolkit, you can also create device-specific content. This involves using the *DeviceSpecific* control within your application and then applying different filters depending on the connecting device. For example, if an HTML-enabled browser is connecting, images can be returned in a GIF or JPEG format. However, these image types are not supported in WAP applications; hence, a WBMP image would be returned to WAP-enabled devices.

Following are the default device filters found in the Web.Config file and used for the delivery of device-specific content. As you can see, HTML, WML, and CHTML are supported, and the Mobile Internet Toolkit looks for various arguments in the HTTP header information of connecting devices:

```
    <deviceFilters>
  <!– Markup Languages –>
  <filter name="isHTML32" compare="preferredRenderingType"
      argument="html32" />
  <filter name="isWML11" compare="preferredRenderingType"
      argument="wml11" />
  <filter name="isCHTML10" compare="preferredRenderingType"
      argument="chtml10" />
  <!– Device Browsers –>
  <filter name="isGoAmerica" compare="browser" argument="Go.Web" />
  <filter name="isMME" compare="browser" argument="Microsoft Mobile
       Explorer" />
  <filter name="isMyPalm" compare="browser" argument="MyPalm" />
  <filter name="isPocketIE" compare="browser" argument="Pocket IE" />
  <filter name="isUP3x" compare="type" argument="Phone.com 3.x Browser" />
  <filter name="isUP4x" compare="type" argument="Phone.com 4.x Browser" />

  <!– Specific Devices –>
  <filter name="isEricssonR380" compare="type" argument="Ericsson R380" />
```

```
<filter name="isNokia7110" compare="type" argument="Nokia 7110" />
<!— Device Capabilities —>
<filter name="prefersGIF" compare="preferredImageMIME"
    argument="image/gif" />
<filter name="prefersWBMP" compare="preferredImageMIME"
    argument="image/vnd.wap.wbmp" />
<filter name="supportsColor" compare="isColor" argument="true" />
<filter name="supportsCookies" compare="cookies" argument="true" />
<filter name="supportsJavaScript" compare="javascript" argument="true" />
<filter name="supportsVoiceCalls" compare="canInitiateVoiceCall"
    argument="true" />
 </deviceFilters>
```

You can use various customization techniques to change any number of the characteristics of a Mobile Internet Toolkit application. These can include stylesheets, templates, and property overrides:

- **Stylesheets** Used automatically within the mobile Internet application to render content appropriately. In addition to the automatic styles that exist within the Mobile Internet Toolkit, you can also use external style element to customize application content.
- **Templates** Used to customize the layout of Mobile Controls in a Mobile Web Form. They can contain multiple controls and even code fragments; used correctly, they can give your application a consistent look and feel.
- **Property overrides** Allow you to change certain properties of Mobile Controls and application functionality.

Using Custom Controls

In addition to the built-in Mobile Controls that come with the Mobile Internet Toolkit, you can also create custom controls for use within your Mobile Internet Toolkit application. These can be Mobile User Controls, which look like any Mobile Web Form element, or they can be Mobile Custom Controls allowing you to define and customize content in some way.

Mobile User Controls involve creating an ASCX file containing any combination of mobile controls and content. You can use these Mobile User Controls within the application in the same way as regular mobile controls. When using custom mobile user controls, you must register the custom tag prefixes in the @Register directive. Because custom Mobile User Controls are based on regular Mobile Controls, the existing device adaptors are used for rendering content to mobile devices.

The other types of custom controls are the Mobile Custom Controls. These type of controls are written and compiled separately, can be written with any CLR language, and provide the maximum flexibility. There are three types of Mobile Custom Controls:

- **Composition controls** Combination of multiple existing controls.
- **Inheritance controls** Extends an existing control in some way.
- **Direct controls** New control from scratch.

Mobile Information Server

As part of the Microsoft mobile solutions offering, Mobile Information Server (MIS) is positioned to be an important piece of an overall mobile solution. MIS is a platform for the delivery of mobile applications. Version 1.0 was released in summer 2001. MIS comes in an Enterprise and Carrier edition providing an end-to-end platform for the delivery of enterprise mobile solutions. MIS typically is hosted in the DMZ (Demilitarized Zone) and sits between enterprise data sources and the various wireless devices. This platform provides an integrated environment for the delivery of mobile applications including enhanced security mechanisms and direct backend integration with the Active Directory. You can use MIS Enterprise Edition for securely deploying Mobile Internet Toolkit applications as well as pushing notifications generated in a Mobile Internet Toolkit event or any other enterprise event sources.

One feature that comes as part of the Mobile Information Server platform is Intranet Browse. Intranet Browse functionality allows access to intranet-hosted WAP applications. Once you have developed, tested, and deployed your Mobile Controls application with the Mobile Internet Toolkit, you can access these applications using Mobile Information Server. You can host the application on an intranet Web server. The application would then be accessed with the following URL format using Mobile Information Server: http://<mmisserver>/in/<intranetserver>/<mmitapp>.

Mobile Information Server also features a number of APIs for sending notification. Within your Mobile Internet Toolkit application, if you would like to notify another mobile user of a next step or any other event, you can send notifications securely through Mobile Information Server. These messaging interfaces include COM and SOAP APIs to Mobile Information Server.

Considering Cookieless State Management

Another area of consideration when implementing mobile Internet applications is state management. Typically, in a Web application, state management is done via Cookies or Session variables. However, in the mobile world, support for cookies and session variables can be inconsistent. This is primarily because this information is not stored on the device but instead is maintained by carrier WAP and wireless gateways. For this reason, it is often important to implement cookieless state management.

Hidden variables are the foundation of cookieless state management, however, the Mobile Internet Toolkit does not have a Mobile Control for specifying hidden variables. Within the *MobilePage* class is a collection called *HiddenVariables* that you can use to store session details. When a Mobile Web Form is submitted, variables are automatically populated into this collection.

Summary

Mobile Internet applications will be an important part of enterprise and consumer mobile solutions in the coming years. The Gartner Group predicts that by 2004, 65 percent of Global 2000 companies will have wireless e-mail and other enterprise mobile solutions. With these trends and the broad interest in mobile solutions that we are seeing in the market, as a developer you need to start learning these technologies, and as a company, it is even more important in getting started leveraging mobile solutions.

The mobile Internet industry is new and fragmented, and as we have covered, a number of challenges and considerations exist, such as multiple carriers, wireless protocol, and device types. With these considerations, implementing broad-reaching mobile solutions that function as desired across multiple devices is often very difficult.

The Mobile Internet Toolkit is a solid platform for the delivery of mobile Internet solutions. The rapid delivery of mobile Internet applications targeted to multiple device types is a significant advantage over other mobile solutions. By using Mobile Web Forms and Mobile Controls, you can rapidly implement applications. The integration of the Mobile Internet Toolkit, the Mobile Designer, and Visual Studio.NET provide a solid platform for the development, testing, and deployment of mobile Internet applications.

Extensibility is a significant consideration when implementing mobile Internet solutions. The Mobile Internet Toolkit has the ability to be extended with new device adaptors and custom controls enabling the support of current specialized devices and future device support. Mobile Internet Toolkit will grow with the mobile Internet industry.

The combination of the Mobile Internet Toolkit and Mobile Information Server provides a complete solution for developing and delivery mobile solutions.

Solutions Fast Track

Obtaining and Installing the Microsoft Mobile Internet Toolkit

☑ The implementation of the Mobile Internet Toolkit involves a separate installation on top of the .NET Framework. This is true both for development and production machines, with some additional installs for the development environment when using Visual Studio.NET:

- For production servers, install Windows 2000 and .NET Framework or .NET Server.
- For development servers, install Windows 2000 Professional and .NET Framework or Windows XP with .NET Framework followed by Visual Studio.NET and appropriate programming language support.
- Install Mobile Internet Toolkit in production and development environments.

☑ Within the Mobile Internet Toolkit, support for the following device types is available in version one: WAP, IMODE, Pocket PC (including Pocket PC 2002), PALM, RIM Blackberry, emulators, and standard desktop Internet Explorer versions.

Using Mobile Web Forms

☑ Mobile Web Forms are the foundation of developing a mobile Internet application with the Mobile Internet Toolkit, and they significantly simplify the development of mobile Internet applications.

☑ Importing the Mobile Web Controls into your .aspx page is done using the import and register page directives. The tag prefix is user defined and the Namespace is *System.Web.UI.MobileControls*. Within the Page directive, you can see the language and Codebehind properties that define the default language and the code source connected to the execution of the page.

Using Mobile Controls

☑ Using Visual Studio .NET you can drag and drop appropriate controls for UI and application logic.

☑ Set appropriate properties and parameters for the various mobile controls.

☑ Test output of mobile controls in target devices.

Exploring Advanced Topics

- ☑ Create device-specific content such as different images for target device types.
- ☑ Create custom controls to implement unique application functionality or to simplify development across multiple Mobile Web Forms.
- ☑ Integrate your Mobile Internet Toolkit application with Mobile Information Server.
- ☑ Test the solution thoroughly with relevant device emulators and target devices.
- ☑ Deploy the solution to production Mobile Internet Toolkit server.
- ☑ Access the application from your mobile Internet device using the appropriate URL.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: How does the Mobile Internet Toolkit deliver customized content?

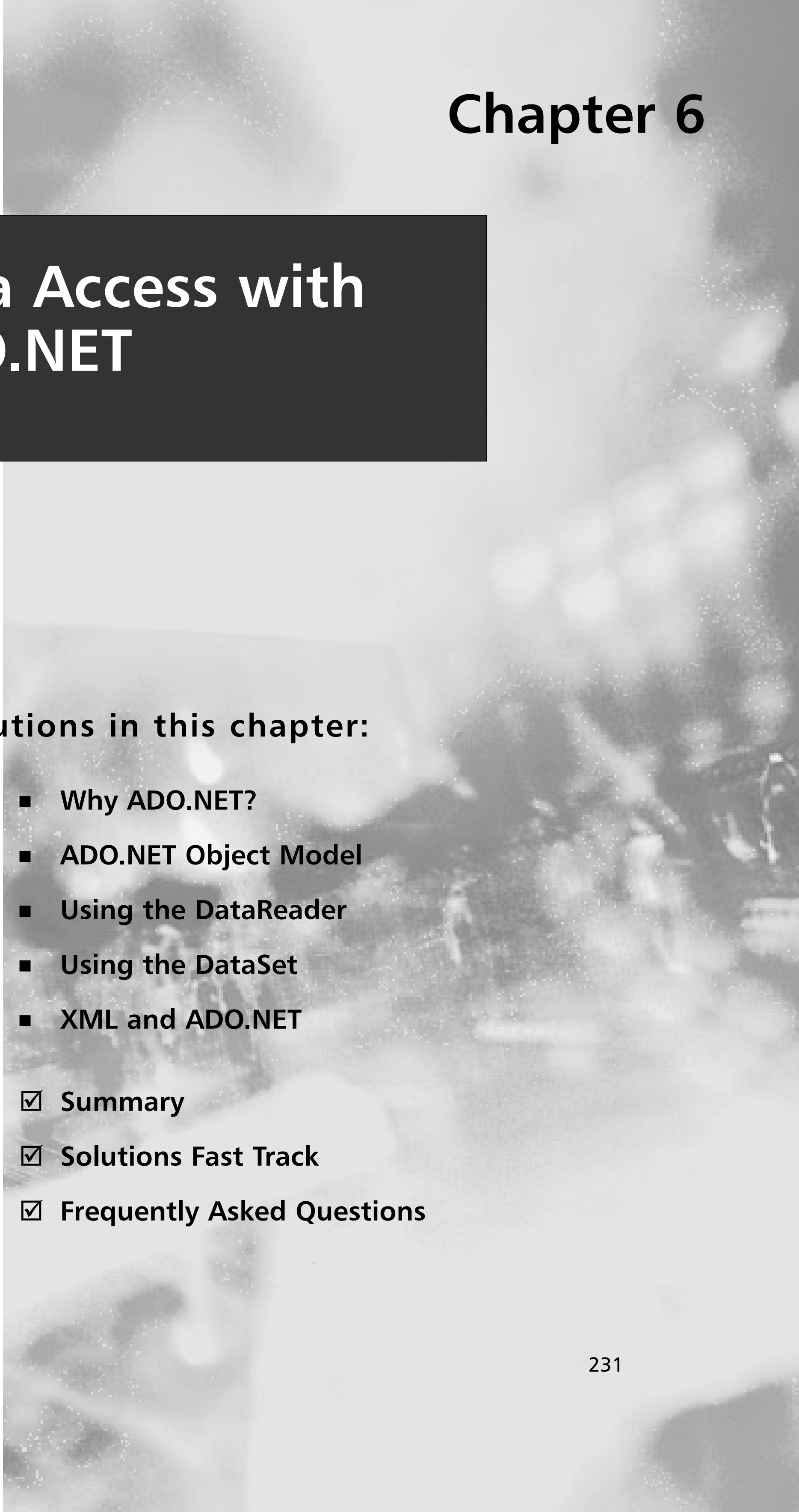
A: The delivery of customized content can be used for different devices and device characteristics. Every device that has a browser sends specific header information when making HTTP requests to the server. For instance, the Motorola i85s (available from Nextel in the U.S.) would identify its browser type as "Phone.com 4.x Browser." Devices can identify themselves as a specific device type, browser type, or supported languages. There are building device filters as part of the Mobile Internet Toolkit, additional specific device filters can also be added to identify unique device types.

Q: How do I support other markup languages and add new device adapters to Mobile Internet Toolkit?

A: As part of the extensibility of the Mobile Internet Toolkit, you can create new adapters to support devices with new characteristics or new markup languages, such as XHTML. The creation of new device adapters is not a trivial process. It involves creating specific adapters for individual device functionality required in your device. For instance, if you want your XHTML-capable device to use a Mobile Internet Toolkit application and take advantage of specific parts of the XHTML language (such as OTAP Over The Air Provisioning), your device adapter would programmatically do this. First, you would create a device filter that would recognize this new device type; the filter would be added to the <deviceFilters> section on the Web.Config file and would reference the device filter source. You would then create the device filter logic, which would be executed when that specific device is detected.

Q: How can I debug Mobile Internet Toolkit applications?

A: Mobile Internet Toolkit errors are handled in exactly the same way as any ASP.NET errors. Errors can occur for a number of reasons—the running of a mobile Web form, the parsing of specific controls, HTTP-intrinsic errors such as 404 file not found, as well as errors reading configuration files or system-specific resource issues. When developing Mobile Internet Toolkit applications, you can use the error reporting mode to allow for debugging of the application. This mode is independent of Visual Studio.NET and provides rich error messages to be display allowing for coding modifications to be determined and made. In addition, as with any ASP.NET application, you can work in Debug mode within Visual Studio.NET. This allows you to step through the code and identify exactly where errors are occurring.



Chapter 6

Data Access with ADO.NET

Solutions in this chapter:

- Why ADO.NET?
- ADO.NET Object Model
- Using the DataReader
- Using the DataSet
- XML and ADO.NET

☑ Summary

☑ Solutions Fast Track

☑ Frequently Asked Questions

Introduction

Microsoft has introduced many kinds of database access technologies. I think everyone remembers Data Access Objects (DAO) or, at least had some dealings with it in some form to read and update an Access database or ODBC-based data source. If Microsoft DAO wasn't your thing, you might have used Open Database Connectivity (ODBC). ODBC technology used an Application Programming Interface (API). Remote Data Objects (RDO) was then introduced. RDO was a thin layer that sat on top of ODBC to provide a more efficient way of accessing data on a server-based DBMS. When the Internet took off, there was a need to take advantage of this medium to retrieve information. In 1996, Microsoft introduced Active Server Pages (ASP) in order to create dynamic Web pages instead of static Web pages. The Web pages were dynamic in that the page itself was able to display information as it was stored in a database at the time the information was retrieved. Traditionally, ASP pages have used the high-level interface called Active Data Objects (ADO) to access databases, because ADO is scriptable. ADO is built on top of the low-level OleDB interfaces.

Microsoft re-examined how their customers accessed and consumed data, how the Internet has changed, and how new technology has come into play (such as XML). Microsoft has envisioned that the .NET Framework will go beyond today's traditional ways of processing information. Microsoft is not looking just to integrate or provide the tools and technologies to solve business requirements at the department level. It is looking at the .NET Framework to solve business requirements at the Enterprise level and between business entities. So, in order to support achieving this goal, a new way of accessing data was needed, and thus ADO.NET was born.

Why ADO.NET?

ADO provided many features. So, why is a new data access model necessary? ADO was a carryover from the other data access models (DAO, ODBC, RDO). The previous data access models were designed around two-tier and client/server development. ADO served well in a COM-based scenario. A managed .NET application could use ADO through COM interoperability, but this solutions is not optimal.

ADO.NET strongly supports the Microsoft .NET Framework and its vision. Microsoft saw the way businesses accessed and consumed information. They saw that what was needed was a more purposeful object model. So, instead of trying to fix ADO for .NET, ADO.NET was built from the ground up. The .NET extension was not just slapped onto ADO to get ADO.NET. Microsoft took the best of ADO and built ADO.NET, keeping in mind the Internet and scalability.

One of the ways that ADO.NET strongly supports the .NET Framework is that ADO.NET knows XML. ADO.NET *DataSet*s (discussed later) are persisted and transmitted over the wire as XML and are transmitted as XML. Transmitting data as XML guarantees interoperability. It guarantees interoperability because XML is an industry standard format. The receiving component does not need to be COM-based. The disparate platform that is receiving the data just needs an XML parser. The receiving component does not have the architectural restrictions that were inherent with COM. As long as the sending and receiving components agree upon the structure (schema) of the data, they can easily share information.

Transmission of an ADO.NET *DataSet* over HTTP is now easier. While ADO *RecordSets* do support an XML format, which is critical for the transmission of data through firewalls, it is a rather obscure flavor of XML. *DataSets* support the W3C standard XML schema format and provide more flexible mapping of the XML into and out of the *DataSet*.

The concept of disconnected Recordsets is nothing new. ADO 2.0 introduced disconnected Recordsets, which made ADO scalable. Using disconnected Recordsets helps conserve system resources by using only a connection to a data store (mostly databases) and having it open for only the amount of time required to extract the data. The connection is opened; the data is extracted; the connection is closed. The connection is open for only a short period of time. ADO.NET *DataSet*s is also scalable in the sense that *DataSet*s are disconnected. Unlike ADO.NET *DataSet*s, ADO Recordsets needed to be programmatically disconnected. This is what makes ADO.NET *DataSet*s more efficient in sharing data and inherently provides a better scalable solution. Bypassing the COM marshalling enables components to use whatever data type is required. This also eliminates the need to do data type conversions in order to comply with COM data types, which increases performance.

ADO.NET Object Model

In the .NET Framework, the .NET Data Provider is the core object that is used to access data. You can use two .NET Data Providers for accessing data: Microsoft SQL Server .NET Provider and the OLE DB .NET Data Provider. The Microsoft SQL Server .NET Data Provider, as you may have guessed, is the provider for accessing data in a Microsoft SQL Server database (MS SQL Server 7.0 and above). The OLE DB .NET Data Provider is used for accessing data exposed through OLE DB, such as ORACLE, DB2, and so on.

As you look at the object model for ADO.NET (see Figure 6.1), it should look fairly familiar. You have the *Connection* object, which is akin to the *Connection* object in ADO to make a connection to a database. Next, you have the *Command* object. Again, this is akin to the *Command* object in ADO where you use it to reference the SQL statement or stored procedure to execute.

Here is where it deviates from ADO. The *DataReader* object is new. This object provides a way to retrieve information. We discuss this object in the next section. The *DataAdapter* contains four *Command* objects (*SelectCommand*, *InsertCommand*, *UpdateCommand*, and *DeleteCommand*). And then there's the *DataSet*. We will discuss the *DataAdapter* and *DataSet* objects later in this chapter.

Let's take a quick look at using the Microsoft SQL Server .NET Data Provider in Figure 6.2 (UpdateCompanyName.aspx). The code is a simple example, which illustrates how you connect to a SQL Server database and use the *Command* object to update a row in the database. Using the *ExecuteNonQuery* method of the *Command* object assumes that you do not expect rows of data to be returned from the call. You can find this source in Figure 6.2 at **www.syngress.com/solutions** in the UpdateCompanyName.aspx file under the Ch6WebApp project.

Figure 6.1 Data Access Object Model of ADO.NET

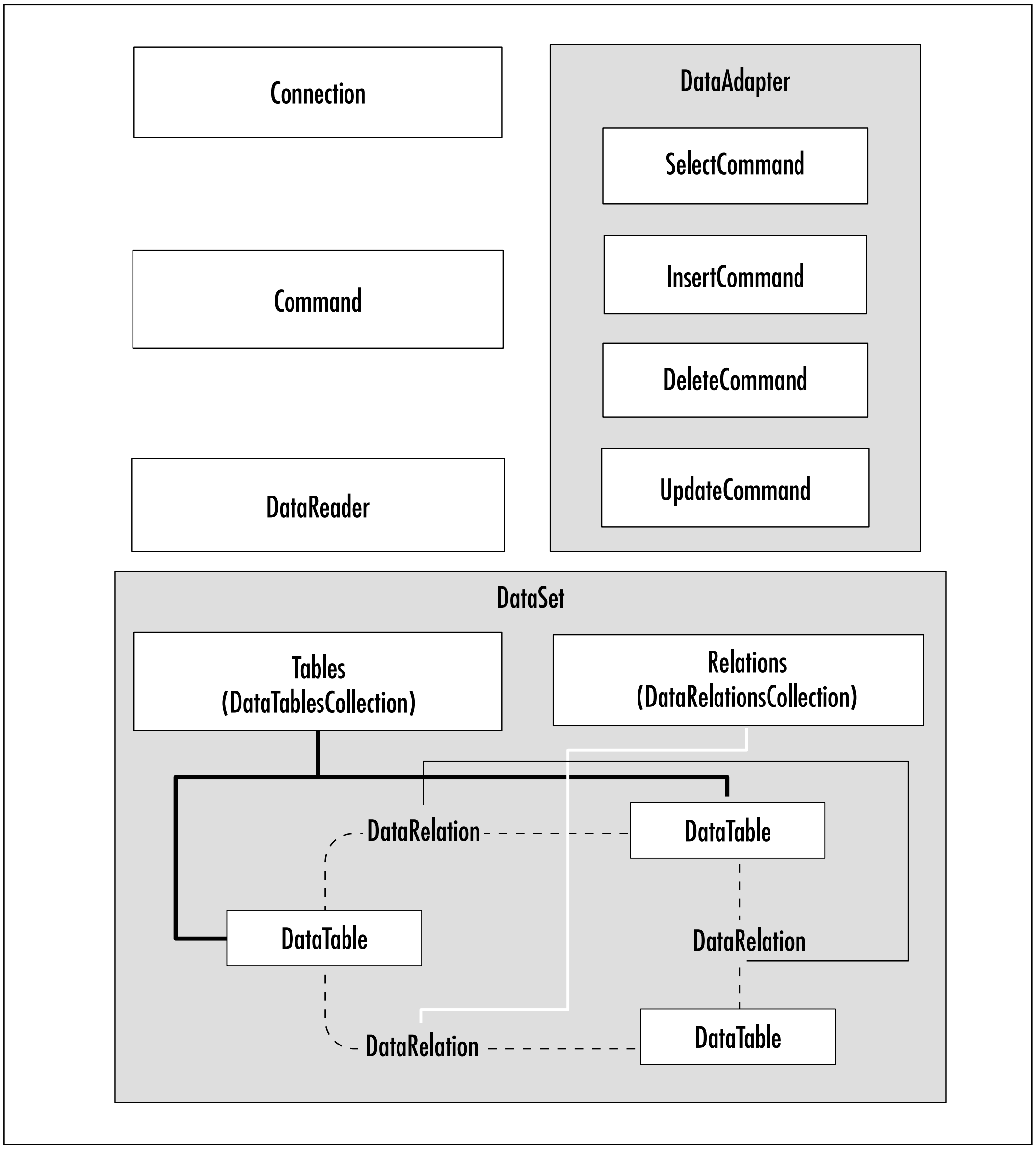


Figure 6.2 ADO.NET Example

```
Private Sub UpdateCompanyName()
    'This creates a new connection object and sets the connection
    Dim oConn As New SqlConnection("Password=a62b34gs;User ID=sa;" & _
```

Continued

Figure 6.2 Continued

```
                "Initial Catalog=Northwind;Data Source=(local)")
    'This creates a new command object
    Dim oCmd1 As New SqlCommand()
    Dim sSQL As String

    sSQL = "Update Customers "
    sSQL = String.Concat(sSQL, "Set CompanyName = '")
    sSQL = String.Concat(sSQL, txtCompanyName.Text.Trim, "' ")
    sSQL = String.Concat(sSQL, "Where CustomerID = '")
    sSQL = String.Concat(sSQL, DDList1.SelectedItem.Value, "'")

    Try
        oConn.Open()
        Label1.Text = sSQL
        With oCmd1
            .CommandType = CommandType.Text
            .CommandText = sSQL
            .Connection = oConn
            .ExecuteNonQuery()
        End With
    Catch oErr As System.Exception
        Label1.Text = "ExecuteNonQuery: " & oErr.Message
    End Try

    'Check on Connection Object, Close it
    If oConn.State = oConn.State.Open Then
        oConn.Close()
    End If

  End Sub
```

This code executes a SQL statement, but does not return any results. The code creates the connection and command object, make the connection to SQL Server, builds your SQL *update* statement, and executes the statement using the

ExecuteNonQuery method. It is pretty reminiscent of ADO, except for the *ExecuteNonQuery* (just *Execute* in ADO). Table 6.1 shows the namespaces of the objects of the ADO.NET classes that provide data access services to applications using Microsoft SQL Server or other data sources. The parent namespace, *System.Data*, contains the interface definitions IDBCommand, IDBConnection, IDataReader, IDataRecord, which are common to all data providers, as well as the entire *DataSet* classes, DataTable, DataRow, DataColumn, DataRelation, and so on. In this chapter, the examples revolve around using Microsoft SQL Server as the backend database. And thus, you will be using the *System.Data.SqlClient* namespace for the Microsoft SQL Server .NET Data Provider. You can easily translate the code examples to use the OLE DB .NET Data Provider. You can still use the OLE DB .NET Data Provider to go against Microsoft SQL Server. However, the Microsoft SQL Server .NET Data Provider is optimized for Microsoft SQL Server 7.0 and later. With the Microsoft SQL Server .NET Data Provider, you don't go through an OleDB layer. The Data Provider directly uses SQL Server tabular data stream (TDS).

Table 6.1 Data Access Services Namespaces

Namespace	Object
System.Data.SqlClient	*SqlConnection, SqlCommand, SqlDataReader, SqlDataAdapter*
System.Data.OleDB	*OleDBConnection, OleDBCommand, OleDBDataReader, OleDBDataAdapter*

When you create a project through Visual Studio.NET, the parent namespace, *System.Data*, is automatically part of the project's references. The code in this chapter uses a shortcut method to easily refer to objects in a namespace without having to fully qualify the parent namespace when defining a variable. This shortcut method is the *Imports* statement. Remember, however, that if conflicts exists (some parent namespaces may have the same names for the underlying objects or methods), you may need to fully qualify your definitions. Take a look at the following example. This example shows how you can use the *Imports* statement to easily reference objects within a parent namespace. The first line in the subroutine defines the *oCmd1* variable using the fully qualified reference to the SQL command object, *System.Data.SQLClient.SqlCommand*. The second line uses the shortcut method to type the variable, because the *Imports* statement was used:

```
Imports System.Data.SQLClient
```

```
Private Sub UpdateCompanyName()
     Dim oCmd1 As New System.Data.SQLClient.SqlCommand()
     Dim oCmd1 As New SqlCommand()

    'code to execute

End Sub
```

Using the DataReader

In this section, you will explore the *DataReader* object to pull data from a datasource. The *DataReader* is a high-performance data access mechanism. What makes this data access model a high-performance data access? The *DataReader* is a read-only and a forward-only database access object. It is also noncached. You also must remember that it is connected. So, what does this all mean? It means that when using the *DataReader*, you are connected to the database; you can read the records from first to last only (no going back to the previous record). And so, once you read a record and move off that record, the record is no longer available. Not much overhead is associated with the *DataReader* object, which is one of the reasons that make it a high-performance data access. It is very useful in retrieving a list of data for populating controls such as list boxes, whether within a Web application or a Windows application.

Figure 6.3 is a simple example illustrating the use of the *DataReader* object. This code shows how to use the *DataReader* to populate two DropDownList boxes on an ASP.NET page. You can find this source in Figure 6.3 at **www.syngress.com/solutions** in LoadDropDowns.aspx file under the Ch6WebApp project.

Figure 6.3 ***DataReader*** **Example**

```
Private Sub LoadDropDowns()
     'Datareader example
     Dim oDR1 As SqlDataReader
     Dim oDR2 As SqlDataReader
     'This creates a new connection object and sets the connection
     Dim oConn As New SqlConnection("Password=a62b34gs;User ID=sa;" & _
                  "Initial Catalog=Northwind;Data Source=(local)")
```

Continued

Figure 6.3 Continued

```
        'This creates a new command object
        Dim oCmd1 As New SqlCommand("Select * from customers", oConn)
        Dim oCmd2 As New SqlCommand("Select * from employees", oConn)
        Dim sName As String
        Dim oItem As ListItem
        Dim sKey As String

        Try
            'Open the Connection to the Database
            oConn.Open()
            Try
                'Execute the DataReader Command
                oDR1 = oCmd1.ExecuteReader(CommandBehavior.SingleResult)
                DDList1.Items.Clear()
                While (oDR1.Read)
                    sName = oDR1.Item("companyname")
                    sKey = oDR1.Item("customerid").ToString
                    'For each row we add the CompanyName to the ListBox
                    oItem = New ListItem(sName, sKey)
                    DDList1.Items.Add(oItem)
                End While
                oDR1.Close()
            Catch oErr As System.Exception
                Label1.Text = "Execute 1: " & oErr.Message
            End Try
            'Execute the DataReader Command
            Try
                oDR2 = oCmd2.ExecuteReader(CommandBehavior.SingleResult)
                DDList2.Items.Clear()
                While (oDR2.Read)
                    'For each row we add the employee to the ListBox
                    sName = oDR2.Item("LastName")
                    sName = sName & ", " & oDR2.Item("FirstName")
                    sName = sName & oDR2.Item("FirstName")
```

Continued

Figure 6.3 Continued

```
                    sKey = oDR2.Item("employeeID").ToString
                    oItem = New ListItem(sName, sKey)
                    DDList2.Items.Add(oItem)
                End While
                oDR2.Close()
            Catch oErr As System.Exception
                Label1.Text = "Execute 2: " & oErr.Message
            End Try
        Catch oErr As System.Exception
            Label1.Text = "SQL Open: " & oErr.Message
        End Try

        'Check on Connection Object, Close it
        If oConn.State = oConn.State.Open Then
            oConn.Close()
        End If

    End Sub
```

As you can see from Figure 6.3, the code for using the *DataReader* is pretty much straightforward. You create the *Connection* object. You then create the *Command* object. After creating the *Command* object, the *DataReader* object is created by calling the *ExecuteReader* method on the *Command* object. You should also notice that the key method of the *DataReader* is the *Read* method. The *Read* method returns either a true or false depending on whether there are more rows to retrieve. The *Read* method will automatically point to the next row. You don't have a *MoveNext* as in ADO. When you compare the *DataReader* code with code using ADO, the *DataReader* code is much more simplified. In the code example, notice that the *DataReader* object is a *SqlDataReader* object. ADO.NET has two types of *DataReader* objects: *OleDbDataReader* and *SqlDataReader*.

A couple of things to remember:

- Always call the *Close* method when you are done with the *DataReader* object.
- Always close the database connection.

If you forget to use the *Close* method or close the database connection, the *Garbage Collector* will do that for you. However, you shouldn't rely on the Garbage Collector. The Garbage Collector will do them on its own time when it gets around to it. In the meantime, valuable resources are being taking up. In the *DataReader* code example, you are populating two drop-down list boxes. If you comment out the call to the *Close* method of the first *DataReader* object and then run the application, what you'll notice is that an exception will occur. Because the first *DataReader* object was not closed, it is keeping a hold on the database connection. The second *DataReader* cannot use the database connection until the first *DataReader* object lets go of it. If you are expecting return values or output parameters from your *Command* object, these are made available to you after the *DataReader* object is closed. The reasons to make sure that you close the database connection when you are done with it and not relying on the Garbage Collector is fairly obvious. Until the Garbage Collector comes around and removes the connection, the database connection may still be open, which is a waste of resources and a drain on the database.

In the code, you used the parameter *CommandBehavior.SingleResult* on the *ExecuteReader* method to return rows of data and explicitly coded the *Close* method of the *Connection* object. Instead of using the *SingleResult* parameter and explicitly coding the *Close* method, you could have easily used the parameter *CommandBehavior.CloseConnection* to have the *DataReader* object close the connection once the *DataReader* object was closed.

Using the DataSet

Now, let's take a look at the *DataSet*. Unlike the *DataReader* object, the *DataSet* object is an in-memory cache of data. This means that you have the ability to move forwards and backwards through the data. The *DataSet* differs from a Recordset in ADO in that the *DataSet* is truly disconnected. The Recordset in ADO had to be programmatically disconnected from the data source. Even though you don't see it in code, the *DataReader* object is used to retrieve the data to be stored in the *DataSet* object. Looking back at the ADO.NET Object Model, the *DataSet* is made up of a collection of objects (*DataTable*s and *DataRelations*).

The *DataTable* object represents a table of in-memory data. The *DataTable* is also made up of collections. The *DataTable* contains a collection of columns (*DataColumn* objects), a collection of rows (*DataRow* objects), and a collection of constraints (*Constraint* objects). A *DataRelation* is used to create parent-child relationships between *DataTable* objects. The *DataRelation* between *DataTables* ensures referential integrity within the *DataSet*. The *DataRow* and *DataColumn* objects are self-describing. The *Constraint* objects ensure data integrity within your *DataTable*. Once the relationships have been created, you can use the *DataRelation* object to return the child or parent rows of a particular row.

Let's take a quick look at using a *DataSet* (see Figure 6.4). The code in this figure shows how you can retrieve data using a *DataSet*. Code is available at **www.syngress.com/solutions** in the Ch6WebApp project under the DataSetQuickLook.aspx file.

Figure 6.4 Simple *DataSet* Example

```
Private Sub DataSetQuickLook()
   'This creates a new connection object and sets the connection
   Dim oConn As New SqlConnection("Password=a62b34gs;User ID=sa;" & _
                "Initial Catalog=Northwind;Data Source=(local)")
   'This creates a new command object
   Dim oCmd As New SqlCommand("Select * from customers", oConn)
   Dim oDA As SqlDataAdapter
   Dim oDS As New DataSet()
   Dim iCnt As Int32
   Dim sName As String
   Dim oItem As ListItem
   Dim sKey As String

   'Create the DataAdapter
   oDA = New SqlClient.SqlDataAdapter(oCmd)
   'Fill the DataSet
   oDA.Fill(oDS)
   DDList3.Items.Clear()
   'point to the results stored in the DataTable
   With oDS.Tables(0)
       'Iterate through the rows of the DataSet to fill dropdown
       For iCnt = 0 To .Rows.Count - 1
           sName = .Rows(iCnt).Item("CompanyName").ToString()
           sKey = .Rows(iCnt).Item("CustomerID").ToString
```

Continued

Figure 6.4 Continued

```
            oItem = New ListItem(sName, sKey)
            DDList3.Items.Add(oItem)
        Next
    End With

   End Sub
```

The *DataAdapter* and the *DataSet* work hand-in-hand to retrieve information from a data source and update the information in the data source. The *DataAdapter* is essential to ADO.NET. The *DataAdapter* is the set of objects that lets you exchange data between a data source and a *DataSet*. What this means is that you configure the *DataAdapter* to determine what data needs to move into and out of a *DataSet*. Usually, adapters are configured to use SQL statements and stored procedures to read and write to a database. Adapters are not limited to just databases. You might have an adapter that has, for example, Microsoft Exchange as a data source. Adapters can move data from any data source into and out of a *dataset*. From the *DataAdapter*, the *Fill* method is used to extract the data and store it in the *DataSet* as a *DataTable*.

As in ADO, where you can hand craft your own Recordset, you can also handcraft your own *DataSet*:

```
  Private Function GetCustomDataSet() As DataSet
     Dim oDS As New DataSet("Contacts")
     Dim oDT As DataTable = oDS.Tables.Add("People")
     Dim oRow As DataRow

     With oDT
         .Columns.Add("ContactID", Type.GetType("System.Int32"))
         .Columns.Add("FirstName", Type.GetType("System.String"))
         .Columns.Add("LastName", Type.GetType("System.String"))
         .Columns.Add("PhoneNbr", Type.GetType("System.String"))
     End With

     oRow = oDT.NewRow()
     With oRow
```

```
            .Item("ContactID") = 1
            .Item("FirstName") = "John"
            .Item("LastName") = "Doe"
            .Item("PhoneNbr") = "555-555-5555"
        End With
        oDT.Rows.Add(oRow)

        oRow = oDT.NewRow()
        With oRow
            .Item("ContactID") = 2
            .Item("FirstName") = "Jane"
            .Item("LastName") = "Doe"
            .Item("PhoneNbr") = "555-555-5555"
        End With
        oDT.Rows.Add(oRow)

        Return oDS
    End Function
```

Relationships and Constraints

This section discusses *DataRelations*. With *DataSet*s, you can have multiple results (collection of *DataTable*s) within the same *DataSet*. You were also able to have multiple results in ADO Recordsets. What you couldn't do with a Recordset was relate the multiple resultsets or set up relationships between one or more of the resultsets. As you would in a DBMS such as Microsoft SQL Server, you can create relationships between tables in order to ensure referential integrity among parent-child records.

Here is an example of how you can create relationships between *DataTable*s within a *DataSet*. In the following code, you retrieve records from two database tables, *Customers* and *Orders*, and store the records in a *DataSet* under the corresponding *DataTable*s, *Customers* and *Orders*. you then define the parent-child relationship between the two *DataTable*s based on *CustomerID* and create a constraint:

```
Public Function GetCustomeOrders() As DataSet
  Dim oDA1 As New SqlDataAdapter("Select * from customers", oConn)
  Dim oDA2 As New SqlDataAdapter("Select * from orders", oConn)
```

```
   Dim oDS As New DataSet("CustomerOrders")

   oConn.Open()
   'set up a datatable in dataset with Customers
   oDA1.Fill(oDS, "Customers")
   'set up a datatable in dataset with Customers
   oDA2.Fill(oDS, "Orders")

   'set up relationship between the datatables
   With oDS
     .Relations.Add("CustOrders", _
       .Tables("Customers").Columns("CustomerID"), _
       .Tables("Orders").Columns("CustomerID"), True).Nested = True
   End With

      'Check on Connection Object, Close it
      If oConn.State = oConn.State.Open Then
          oConn.Close()
       End If

      Return oDS

  End Function
```

You've seen how you can use the *Fill* method to populate a *DataTable* in a *DataSet*. The information about each of the columns in the *DataTable* is automatically brought down into the *DataColumnsCollection*. You've seen how you can relate tables to ensure referential integrity. But how can you ensure data integrity within the *DataTable*? There is also a method on the *DataAdapter* that you can invoke to retrieve the constraints (primary keys, foreign keys, unique constraints, and so on) of the *DataTable*. This method is the *FillSchema* method:

```
oDA.FillSchema(oDS, SchemaType.Mapped, "Customers")
```

You can programmatically create the constraints to supply, if not supplement, constraints. We show how this is done in this series of examples.

First, you declare your variables. In the code, you refer to an object, *oDS*, which is a *DataSet* object that has been already created:

```
'declare the parent table
Dim oPTbl As DataTable = oDS.Tables("Customers")
'declare the child table
Dim oCTbl As DataTable = oDS.Tables("Orders")
'declare the parent column for CustomerID
Dim oPColID As DataColumn = oPTbl.Columns("CustomerID")
'declare the parent column for ContactName
Dim oPColCN As DataColumn = oPTbl.Columns("ContactName")
'declare the child column
Dim oCCol As DataColumn = oCTbl.Columns("CustomerID")
```

Here is an example for creating a *UniqueConstraint*.You need to make sure that the column is not able to contain a Null value:

```
'make new UniqueConstraint object,add to Constraints collection
Dim oUnique As New UniqueConstraint("UC_ContactName", oPColCN)
oPTbl.Constraints.Add(oUnique)

'make sure that the column cannot have a Null value
oPColCN.AllowDBNull = False
```

Here is an example of creating a *PrimaryKey* constraint.You create an array to store the column objects that would make up the primary key. In this case, only one column, *CustomerID*, makes up the primary key of the *Customers* table.You may need to make sure that the column or columns cannot contain a Null value, especially if it is a hand-crafted *DataSet:*

```
'create an array to hold more the one column
Dim arrCol(0) As DataColumn

'make sure that the column cannot have a Null value
oPColID.AllowDBNull = False

arrCol(0) = oPCol

'set this array as the columns for the primary key
oPTbl.PrimaryKey = arrCol
```

Lastly, here is an example of creating a *ForeignKeyConstraint*:

```
'create a new ForeignKeyConstraint object
Dim oFKey As New ForeignKeyConstraint("FK_CustomerID", _
  oPColID, oCCol)

'set the "update" properties
oFKey.DeleteRule = Rule.Cascade
oFKey.UpdateRule = Rule.Cascade

CTbl.Constraints.Add(oFKey)
```

You can use the following four options for the "update" rules for the foreign key constraints:

- **Cascade** Updates to the primary key value in the parent table are copied to the foreign key in all linked child rows. Deleting a parent row deletes all linked child records.
- **SetDefault** Updates to the primary key value in the parent table or deletion of a parent row both causes the foreign key in all linked child rows to be set to its default value.
- **SetNull** Is the same as SetDefault except value is Null.
- **None** Updates have no effect.

You can write out the schema of a *DataSet* to a file by using the *WriteXmlSchema* method of the *DataSet* object:

```
oDS.WriteXmlSchema("c:\Customers.xsd").
```

Displaying Data

So now you have a *DataSet*. How do you display it? You can use a *DataGrid* to display the *DataSet*. Each *DataTable* in a *DataSet* has a *DefaultView* property. You can use this property to display the records in a datagrid:

```
Dim wsNorthwind As New localhost.NorthwindData()
Dim oDS As DataSet
oDS = oWS.GetCustomerList
DataGrid1.DataSource = oDS.Tables(0).DefaultView
DataGrid1.DataBind()
```

In this example, you will call a function to retrieve data. The function returns a *DataSet* containing data from the ubiquitous Northwind database. I specified the first table, *Tables(0)*, in the *DataTable*s collection. The collection is zero-based. If the *DataTable* was assigned a table name, you could have referenced the *DataTable* by its name, *Tables("Customers")*. The table name is assigned on the *Fill* method of the *DataAdapter*. In the examples in this chapter, you pretty much reference things by their indexes:

```
oDA.Fill(oDS, "Customers")
```

The *DefaultView* method can also return a *DataView* object. You can use the *DataView* object to display the records in different ways, such as sorted by *ContactName*:

```
Dim c As New localhost.NorthwindData()
Dim oDS As New DataSet()
Dim oDV As DataView

oDS = wsNorthwind.GetCustomerList
oDV = oDS.Tables(0).DefaultView
oDV.Sort = "ContactName asc"
DataGrid1.DataSource = oDV
DataGrid1.DataBind()
```

Of course, you can apply the sorting directly on the *DefaultView* without having to use a separate *DataView* object. The advantage of using a separate *DataView* object is that you can apply the sort criteria to the *DataView* object without affecting the *DefaultView*. The *DefaultView* maintains its own view of the data:

```
oDS.Tables(0).DefaultView.Sort = "ContactName asc"
```

What is sorting without being able to filter records? The *DataView* object in ADO.NET as well as the *DefaultView* property of the *DataTable* object support filtering of records with the *RowFilter* property:

```
oDV.RowFilter = "contactname like 'r%'"
```

Another type of filter is based on *RowState* (discussed later). You can set this in the *RowStateFilter* property for both:

```
oDV.RowStateFilter = DataViewRowState.Deleted
```

The *DataTable* object also makes available the *Select* method, which will sort and filter records, which you can use in the following ways:

- To filter the rows:

  ```
  oDS.Tables(0).Select("contactname like 'r%'")
  ```

- To filter and then sort the rows:

  ```
  oDS.Tables(0).Select("contactname like 'r%'", "ContactName desc")
  ```

- To filter and sort only rows based on *RowState*, which we discuss in the next section:

  ```
  oDS.Tables(0).Select("contactname like 'r%'", "ContactName desc"
                       , _ DataViewRowState.Added)
  ```

- To return an array of *DataRow* objects so you can programmatically spin through the rows:

  ```
     Dim oRows() As DataRow
     oRows = oDS.Tables(0).Select()
  ```

Editing the *DataSet*

Now that you have a *DataSet* and are able to see the information residing in the *DataTable*s, how can you edit the information? You've seen a little bit of how to make changes to the information in previous examples using the *NewRow* method and filling the row items with values. Let's take a closer look at editing the *DataSet* tables. First, we should go over *RowState* and *RowVersion*.

The *RowState* basically says what has been done to that row. This is used by the *DataAdapter* to determine what needs to be done in order to make updates to the data source (the SQL Server database in the examples in this chapter). You'll notice a couple of references to *AcceptChanges*; we discuss these shortly. (It will all fall into place. There isn't a good way to break up some of the topics in a clean-cut way because they overlap.) So, for now, here is the description of the values of the *RowState* property of the *DataRow* object.

Here is the enumeration of the values is *DataRowState*:

- **Added** This row was created and added to the *DataRowCollection*. *AcceptChanges* has not been called.

- **Deleted** This row has been marked for deletion by the *Delete* method of the *DataRow* object.
- **Detached** This row was created, but either it has not been added to the *DataRowCollection*, or it was removed from the *DataRowCollection*.
- **Modified** The contents of this row was modified and the *AcceptChanges* was not called.
- **Unchanged** No changes have been made to this row since the last time that *AcceptChanges* was called.

When editing values for a row, different versions of the row are created in order to provide data concurrency. The enumeration for *RowVersion* is *DataRowVersion* is as follows:

- **Current** This row contains the current values.
- **Default** This row contains the default values of the columns.
- **Original** This row contains the original values when the *DataTable* was filled. The *AcceptChanges* overlays the original values with the values from *Current*. You can use the original values to determine if another user has made updates to the data source since you're retrieval of the data.
- **Proposed** This row version is created when you do a *BeginEdit*, and it contains the changed values.

To find out what row versions exist for a row, you can use the *HasVersion* method of the *DataRow*. The *HasVersion* method returns a Boolean value (true or false). For example:

```
Dim sMsg As String
With oDS.Tables(0).Rows(0)
   sMsg = String.Concat("Has current row version: ", _
       .HasVersion(DataRowVersion.Current).ToString)
   sMsg = String.Concat(sMsg, " Has default row version: ", _
       .HasVersion(DataRowVersion.Default).ToString)
   sMsg = String.Concat(sMsg, " Has original row version: ", _
       .HasVersion(DataRowVersion.Original).ToString)
   sMsg = String.Concat(sMsg, " Has proposed row version: ", _
       .HasVersion(DataRowVersion.Proposed).ToString)
   Label2.Text = sMsg
End With
```

Adding and deleting rows is pretty straightforward. Looking at a previous example, you use the *NewRow* method to create the *DataRow* object and then use the *Add* method of the *DataTable* to add the new row object to the *DataTable*'s *DataRowCollection*:

```
oRow = oDT.NewRow()
With oRow
    .Item("ContactID") = 1
    .Item("FirstName") = "John"
    .Item("LastName") = "Doe"
    .Item("PhoneNbr") = "555-555-5555"
End With
oDT.Rows.Add(oRow)
```

Editing the values for the items in the row is equally straightforward. You basically just assign the value to the item as shown in the example. Deleting rows is also pretty much straightforward. The *DataRow* object has a *Delete* method:

```
oDS.Tables(0).Rows(0).Delete()
```

Using the *Delete* method doesn't necessarily mean that the actual physical row has been deleted. The row is essentially marked for deletion by setting the *RowState* property to Deleted (*DataRowState.Deleted*). So, when is the row physically removed from the *DataTable*'s *DataRowCollection*? This is where *AcceptChanges* comes into play. Invoking *AcceptChanges* makes the changes to the *DataTable* or to all the *DataTable*s in the *DataSet* permanent.

The *DataRow* object exposes methods so that you can decide whether you want to accept the edits made to a row. You can apply some business rules. If the new values do not conform to those business rules, you can discard those new values and go back to the current values. If the new values do conform to your business rules, you can make those new values the current values of the row. The *BeginEdit*, *CancelEdit*, and the *EndEdit* are the methods that help you determine if you want to keep the new values of a row or to discard those new values:

```
Try
    With oDS.Tables(0).Rows(0)
        MessageBox.Show("original: " & _
            .Item(0, DataRowVersion.Original).ToString)
        MessageBox.Show("current: " & _
                    .Item(0, DataRowVersion.Current).ToString)
                .BeginEdit()
                .Item(0) = "NormG"
                MessageBox.Show("original: " & _
                    .Item(0, DataRowVersion.Original).ToString)
                MessageBox.Show("current: " & _
                    .Item(0, DataRowVersion.Current).ToString)
                MessageBox.Show("proposed = " & _
                    .Item(0, DataRowVersion.Proposed).ToString)
                If .Item(0, DataRowVersion.Proposed) = "NormG" Then
                    .CancelEdit()
                Else
                    .EndEdit()
                End If
                MessageBox.Show("original: " & _
                    .Item(0, DataRowVersion.Original).ToString)
                MessageBox.Show("current: " & _
                    .Item(0, DataRowVersion.Current).ToString)
            End With
        Catch objError As Exception
            MessageBox.Show(objError.Message)
        End Try
```

As you can see from the code, you actually have two versions of the row, *original* and *current*. When you call the *BeginEdit* method, another version, *proposed*, of the row is created. So, when you edit an item of the row, the value you assign to the item is actually stored in the proposed version. The values in the original and current version of the row are unchanged. Now, you can apply your business rule to see if you want to keep the new value for the item, modify it even more, or discard it altogether. Notice that on the *IF* statement, you are specifying the row version when interpreting the value to compare. Now, you decide whether to keep it or discard it. When you invoke the *CancelEdit* method, the *proposed* version of the row is discarded. So, any edits on items in that row are now gone. The *original* and *current* row versions are not affected. In essence, you go back to the way it was before you invoked the *BeginEdit* method. Invoking the *EndEdit* method is, obviously, a different story. What happens is that the values from the proposed version are moved to the current version. Thus, it makes the proposed values the current values. The proposed version is no longer available.

So, is there any time the original values get changed? Yes—the original values do get changed at certain times. Obviously, if you get the data again from the data source, and the values in the data source changed, then the original values would be different. Retrieving the information from the data source back into the *DataSet* and into the same *DataTable* wipes what you had there before. *AcceptChanges* and the *RejectChanges* are the methods that determine if the values in the original row version get changed. They also effectively call the *EndEdit* and *CancelEdit* methods, respectively. These methods are available on the *DataRow*, *DataTable*, and *DataSet* objects.

If you make changes to the rows, and you invoke the *AcceptChanges* method, the following takes place:

- The values of the current row version are replaced by the values from the proposed row version.
- The values of the original row version are replaced by the values from the current row version.
- The proposed row version is discarded.

The *RejectChanges* method, as its name implies, rejects the changes. What you need to do is to be careful on which object you are invoking the methods. Invoking either the *AcceptChanges* or *RejectChanges* methods on the *DataRow* object affects only that *DataRow*. If you invoke either method on the *DataTable* object, this affects all the rows in the *DataRowCollection* for that *DataTable*. If you invoke either method on the *DataSet* object, this affects all the *DataTable*s in that *DataSet*.

To further help you possibly apply business rules or validation, or to just have some ability to find out what changes are occurring in a *DataTable* in a *DataSet*, the *DataTable* object has a series of events that you can use:

- **ColumnChanged** When a column's value has been successfully changed.
- **ColumnChanging** When a value has been submitted to a column.
- **RowChanged** When a row in the table has been successfully edited.
- **RowChanging** When a row in the table is changing.
- **RowDeleted** When a row in the table has been deleted.
- **RowDeleting** When a row in the table is marked for deletion.

In Figure 6.5, the *AddHandler* keyword is used to associate an event handler with an event of the *DataTable.Handlers.* It can be added to associate code with events raised by a *DataTable* such as when data in a column is changed. Code is available at **www.syngress.com/solutions** in the Form1.vb file under the Ch6WinApp project.

Figure 6.5 Event Handler Example

```
AddHandler m_oDS.Tables(0).ColumnChanged, _
       New DataColumnChangeEventHandler(AddressOf OnColumnChanged)
AddHandler m_oDS.Tables(0).RowChanged, _
       New DataRowChangeEventHandler(AddressOf OnRowChanged)


Private Shared Sub OnColumnChanged(ByVal sender As Object, _
                       ByVal args As DataColumnChangeEventArgs)
   MessageBox.Show((args.Column.ColumnName & _
           " changed to '" & args.ProposedValue.ToString() & "'"))
End Sub
Private Shared Sub OnRowChanged(ByVal sender As Object, _
                      ByVal args As DataRowChangeEventArgs)
   If args.Action <> DataRowAction.Nothing Then
       'some code here does something
    End If
End Sub
```

The argument object, *DataColumnChangeEventArgs*, being passed into the event handler for a column exposes details of the column and also exposes details of the row. The argument object, *DataRowEventEventArgs*, exposes details of the row and also what is being done to the row, which is the *Action* (*Add*, *Change*, *Delete*, *Nothing*, *Commit*, *Rollback*).

A new feature of ADO.NET is the strongly typed *DataSet*. Here is example code of using a strongly typed *DataSet*:

```
Dim oConn As New SqlConnection("Password=a62b34gs;User ID=sa;" & _
             "Initial Catalog=Northwind;Data Source=(local)")
'This creates a new command object
Dim oCmd As New SqlCommand("Select * from customers", oConn)
Dim oDA As SqlDataAdapter
Dim oDS As New NorthwindCustDS.CustomersDS.dsCustomers()
Dim oRow As NorthwindCustDS.CustomersDS.dsCustomers.CustomersRow

'Create the DataAdapter
oDA = New SqlClient.SqlDataAdapter(oCmd)
'Fill the DataSet
oDA.Fill(oDS, "Customers")

For Each oRow In oDS.Customers
    ListBox1.Items.Add(oRow.CustomerID.ToString)
 Next
```

Notice that what makes the *DataSet* strongly typed is the way the variable is typed (*dsCustomers*). A big advantage of a typed *DataSet* is that the code is easier to read. You can access tables and columns by name. Notice that you are able to access the name of the column of a *DataRow* instead of going through the *Item* collection and supplying the column name. It may not be a big deal for everyone. You need to consider that now you can actually see the columns that are available without having to look them up and find out how they are spelled. The columns show up through IntelliSense. This eliminates a great deal of type errors and headaches during runtime when you find out that the column doesn't exist because you misspelled it. How do you make a strongly typed *DataSet*? The Framework has a tool, XSD.exe, which lets you create a strongly typed *DataSet*. The tool will generate a component for you to use in your project; it requires is a schema:

```
Xsd.exe Customers.xsd /d /l:VB /n:CustomerDS
```

The tool is located in the .NET Framework SDK directory (\Program Files\Microsoft.NET\FrameworkSDK\Bin). In the example command line, you use the tool giving it a schema, Customers.xsd. You instruct it to generate ("/d") a Visual Basic.NET component (creates Customers.vb) class and the namespace should be CustomerDS.

Here is the schema used:

```
<?xml version="1.0" standalone="yes"?>
<xsd:schema id="dsCustomers" targetNamespace="" xmlns=""
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:msdata=
    "urn:schemas-microsoft-com:xml-msdata">
<xsd:element name="dsCustomers" msdata:IsDataSet="true">
 <xsd:complexType>
   <xsd:choice maxOccurs="unbounded">
    <xsd:element name="Customers">
      <xsd:complexType>
        <xsd:sequence>
         <xsd:element name="CustomerID" type="xsd:string"
             minOccurs="0"/>
         <xsd:element name="CompanyName" type="xsd:string"
             minOccurs="0"/>
         <xsd:element name="ContactName" type="xsd:string"
             minOccurs="0"/>
         <xsd:element name="ContactTitle" type="xsd:string"
             minOccurs="0"/>
         <xsd:element name="Address" type="xsd:string" minOccurs="0"/>
         <xsd:element name="City" type="xsd:string" minOccurs="0"/>
         <xsd:element name="Region" type="xsd:string" minOccurs="0"/>
         <xsd:element name="PostalCode" type="xsd:string"
             minOccurs="0"/>
         <xsd:element name="Country" type="xsd:string" minOccurs="0"/>
         <xsd:element name="Phone" type="xsd:string" minOccurs="0"/>
         <xsd:element name="Fax" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
       </xsd:complexType>
     </xsd:element>
    </xsd:choice>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

You then take the generated VB component class and your schema and add it to a VB.NET *ClassLibrary* project. You don't need to use VB.NET. You can generate a C# component class and place it in C# project. After this, you add the files to your project—just build the project and reference the .NET component in the actual application. You can now easily reference the columns of the table without having to know how they are spelled and easily determine what columns are available.

Updating the Data Source

Now that you have a *DataSet*, how do you update the database with the changes in the *DataSet*? In ADO, batch updates were transparent to the programmer. When a batch update was used on a Recordset, temporary stored procedures were created to *Insert*, *Update*, and *Delete* records in the database. Now, in ADO.NET, you have full control of the updates. You can specify the stored procedures, if not the SQL statements to use, in performing the batch updates. One of the biggest benefits of this ability to have control of the updates is that you can specify using an existing stored procedure. As we've previously mentioned, in ADO, temporary stored procedures were created to do the *Insert*s, *Update*s, and *Delete*s. ADO incurred the overhead of having the DBMS create the temporary stored procedures. This is no longer the case in ADO.NET. You can utilize existing, already verified and compiled, stored procedures for performance gains. Plus, you now have the added ability to customize those stored procedures to fit your requirements (to add logic in the stored procedures, to only update certain columns, to use a view in selecting records, and so on).

Updates are controlled through the *DataAdapter* with the use of its four properties, which are in turn objects themselves. These properties are the *SelectCommand*, *InsertCommand*, *UpdateCommand*, and *DeleteCommand*. You have the ability to manage these objects during runtime. Thus, you can manipulate the command object even before it is executed. Figure 6.6 shows an example of controlling the updates. This code is available at **www.syngress.com/solutions** in the NorthwindData.asmx file under the Ch6WebService project.

Figure 6.6 Update *DataSet* Example

```
Public Sub UpdateCustomerList(ByVal oDS As DataSet)
    Dim oConn As New SqlConnection("Password=a62b34gs;User ID=sa;" & _
                 "Initial Catalog=Northwind;Data Source=(local)")
    Dim oCmdSelect As New SqlCommand()
    Dim oCmdInsert As New SqlCommand()
    Dim oCmdUpdate As New SqlCommand()
    Dim oCmdDelete As New SqlCommand()
    Dim oDA As SqlDataAdapter
```

Continued

Figure 6.6 Continued

```
        'Create the DataAdapter
        oDA = New SqlClient.SqlDataAdapter()
      With oDA
          .SelectCommand = oCmdSelect
          .InsertCommand = oCmdInsert
          .UpdateCommand = oCmdUpdate
          .DeleteCommand = oCmdDelete
      End With

        '
        'oCmdSelect
        '
     With oCmdSelect
        .CommandText = "SELECT CustomerID, CompanyName, ContactName," &
_
        " ContactTitle, Address, City, Region," & _
        " PostalCode, Country, Phone, Fax FROM Customers"
        .Connection = oConn
      End With
      '
      'oCmdInsert
      '
     With oCmdInsert
      .CommandType = CommandType.Text
      .CommandText = "INSERT INTO Customers(CustomerID, CompanyName," & _
      " ContactName, ContactTitle, Address, City, Region, PostalCode," & _
      " Country, Phone, Fax) VALUES (@CustomerID, @CompanyName," & _
      " @ContactName, @ContactTitle, @Address, @City, @Region," & _
      " @PostalCode, @Country, @Phone, @Fax)"
           .Connection = oConn
           .Parameters.Add(New SqlParameter("@CustomerID", _
```

Continued

Figure 6.6 Continued

```
                    System.Data.SqlDbType.NChar, 5, _
                    System.Data.ParameterDirection.Input, False, _
                    CType(0, Byte), CType(0, Byte), "CustomerID", _
                    System.Data.DataRowVersion.Current, Nothing))
            .Parameters.Add(New SqlParameter("@CompanyName", _
                    System.Data.SqlDbType.NVarChar, 40, _
                    System.Data.ParameterDirection.Input, False, _
                    CType(0, Byte), CType(0, Byte), "CompanyName", _
                    System.Data.DataRowVersion.Current, Nothing))
            .Parameters.Add(New SqlParameter("@ContactName", _
                        System.Data.SqlDbType.NVarChar, 30, _
                        System.Data.ParameterDirection.Input, True, _
                        CType(0, Byte), CType(0, Byte),
                            "ContactName", _
                        System.Data.DataRowVersion.Current, Nothing))
            .Parameters.Add(New SqlParameter("@ContactTitle", _
                        System.Data.SqlDbType.NVarChar, 30, _
                        System.Data.ParameterDirection.Input, True, _
                        CType(0, Byte), CType(0, Byte),
                            "ContactTitle", _
                        System.Data.DataRowVersion.Current, Nothing))
            .Parameters.Add(New SqlParameter("@Address", _
                        System.Data.SqlDbType.NVarChar, 60, _
                        System.Data.ParameterDirection.Input, True, _
                        CType(0, Byte), CType(0, Byte), "Address", _
                        System.Data.DataRowVersion.Current, Nothing))
            .Parameters.Add(New SqlParameter("@City", _
                        System.Data.SqlDbType.NVarChar, 15, _
                        System.Data.ParameterDirection.Input, True, _
                        CType(0, Byte), CType(0, Byte), "City", _
                        System.Data.DataRowVersion.Current, Nothing))
            .Parameters.Add(New SqlParameter("@Region", _
                        System.Data.SqlDbType.NVarChar, 15, _
```

Continued

Figure 6.6 Continued

```
                        System.Data.ParameterDirection.Input, True, _
                        CType(0, Byte), CType(0, Byte), "Region", _
                        System.Data.DataRowVersion.Current, Nothing))
             .Parameters.Add(New SqlParameter("@PostalCode", _
                        System.Data.SqlDbType.NVarChar, 10, _
                        System.Data.ParameterDirection.Input, True, _
                        CType(0, Byte), CType(0, Byte), "PostalCode",
_
                        System.Data.DataRowVersion.Current, Nothing))
             .Parameters.Add(New SqlParameter("@Country", _
                        System.Data.SqlDbType.NVarChar, 15, _
                        System.Data.ParameterDirection.Input, True, _
                        CType(0, Byte), CType(0, Byte), "Country", _
                        System.Data.DataRowVersion.Current, Nothing))
             .Parameters.Add(New SqlParameter("@Phone", _
                        System.Data.SqlDbType.NVarChar, 24, _
                        System.Data.ParameterDirection.Input, True, _
                        CType(0, Byte), CType(0, Byte), "Phone", _
                        System.Data.DataRowVersion.Current, Nothing))
             .Parameters.Add(New SqlParameter("@Fax", _
                        System.Data.SqlDbType.NVarChar, 24, _
                        System.Data.ParameterDirection.Input, True, _
                        CType(0, Byte), CType(0, Byte), "Fax", _
                        System.Data.DataRowVersion.Current, Nothing))
     End With
     '
     'oCmdUpdate
     '
    With oCmdUpdate
      .CommandType = CommandType.Text
      .CommandText = "UPDATE Customers SET CustomerID = @CustomerID," &_
      " CompanyName = @CompanyName, ContactName = @ContactName," & _
      " ContactTitle = @ContactTitle, Address = @Address, City = @City,"
```

Continued

Figure 6.6 Continued

```
         & _
      " Region = @Region, PostalCode = @PostalCode, Country = @Country,"
         & _
      " Phone = @Phone, Fax=@Fax
WHERE(CustomerID=@Original_CustomerID)"
      .Connection = oConn
      .Parameters.Add(New SqlParameter("@CustomerID", _
               System.Data.SqlDbType.NChar, 5, _
               System.Data.ParameterDirection.Input, False, _
               CType(0, Byte), CType(0, Byte), "CustomerID", _
               System.Data.DataRowVersion.Current, Nothing))
      .Parameters.Add(New SqlParameter("@CompanyName", _
               System.Data.SqlDbType.NVarChar, 40, _
               System.Data.ParameterDirection.Input, False, _
               CType(0, Byte), CType(0, Byte), "CompanyName", _
               System.Data.DataRowVersion.Current, Nothing))
       .Parameters.Add(New SqlParameter("@ContactName", _
                System.Data.SqlDbType.NVarChar, 30, _
                System.Data.ParameterDirection.Input, True, _
                CType(0, Byte), CType(0, Byte), "ContactName", _
                System.Data.DataRowVersion.Current, Nothing))
       .Parameters.Add(New SqlParameter("@ContactTitle", _
                System.Data.SqlDbType.NVarChar, 30, _
                System.Data.ParameterDirection.Input, True, _
                CType(0, Byte), CType(0, Byte), "ContactTitle", _
                System.Data.DataRowVersion.Current, Nothing))
       .Parameters.Add(New SqlParameter("@Address", _
                 System.Data.SqlDbType.NVarChar, 60, _
                 System.Data.ParameterDirection.Input, True, _
                 CType(0, Byte), CType(0, Byte), "Address", _
                 System.Data.DataRowVersion.Current, Nothing))
      .Parameters.Add(New SqlParameter("@City", _
```

Continued

Figure 6.6 Continued

```
                  System.Data.SqlDbType.NVarChar, 15, _
                  System.Data.ParameterDirection.Input, True, _
                  CType(0, Byte), CType(0, Byte), "City", _
                  System.Data.DataRowVersion.Current, Nothing))
         .Parameters.Add(New SqlParameter("@Region", _
                  System.Data.SqlDbType.NVarChar, 15, _
                  System.Data.ParameterDirection.Input, True, _
                  CType(0, Byte), CType(0, Byte), "Region", _
                  System.Data.DataRowVersion.Current, Nothing))
         .Parameters.Add(New SqlParameter("@PostalCode", _
                  System.Data.SqlDbType.NVarChar, 10, _
                  System.Data.ParameterDirection.Input, True, _
                  CType(0, Byte), CType(0, Byte), "PostalCode", _
                  System.Data.DataRowVersion.Current, Nothing))
         .Parameters.Add(New SqlParameter("@Country", _
                  System.Data.SqlDbType.NVarChar, 15, _
                  System.Data.ParameterDirection.Input, True, _
                  CType(0, Byte), CType(0, Byte), "Country", _
                  System.Data.DataRowVersion.Current, Nothing))
         .Parameters.Add(New SqlParameter("@Phone", _
                  System.Data.SqlDbType.NVarChar, 24, _
                  System.Data.ParameterDirection.Input, True, _
                  CType(0, Byte), CType(0, Byte), "Phone", _
                  System.Data.DataRowVersion.Current, Nothing))
         .Parameters.Add(New SqlParameter("@Fax", _
                  System.Data.SqlDbType.NVarChar, 24, _
                  System.Data.ParameterDirection.Input, True, _
                  CType(0, Byte), CType(0, Byte), "Fax", _
                  System.Data.DataRowVersion.Current, Nothing))
         .Parameters.Add(New SqlParameter("@Original_CustomerID", _
                  System.Data.SqlDbType.NChar, 5, _
                  System.Data.ParameterDirection.Input, False, _
                  CType(0, Byte), CType(0, Byte), "CustomerID", _
```

Continued

Figure 6.6 Continued

```
                  System.Data.DataRowVersion.Original, Nothing))
        End With
        .
        .
        oDA.Update(oDS)

    End Sub
```

In the code, you created *Command* objects for each type of database access (*Select*, *Insert*, *Update*, and *Delete*). For each command object, you have specified the appropriate command type, SQL statement, connection, and parameters. You have also assigned the command objects to the appropriate *DataAdapter* property. (We did not include all of the code in the listing [*Update* and *Delete*]). Now you are able to call the *Update* method. The *DataAdapter* will use the commands that you have specified for the appropriate action.

You could have easily used stored procedures instead of using a SQL statement by making a couple of minor changes. Instead of saying it is a *CommandType* of *Text*, it is a *CommandType* of *StoredProcedure*. Instead of a SQL statement as the *CommandText*, it is the name of a stored procedure defined in the database. For example:

```
        With oCmdSelect
            .CommandType = CommandType.StoredProcedure
            .CommandText = "spGetCustomerList"
            .Connection = oConn
        End With
```

Just like in ADO, if the *InsertCommand*, *UpdateCommand*, and *DeleteCommand* are not specified, the adapter will automatically generate the SQL statements at runtime. This doesn't give you the most optimal performance because this, just like in ADO, incurs overhead.

As you can see from the code, the command objects for selecting, inserting, updating, and deleting customer records is configured for the adapter. We mentioned earlier that the *DataSet* can contain multiple *DataTables*. How should you handle this particular situation when you have multiple tables to update? You need to create a separate *DataAdapter* for each of the tables that need to be updated. And for each of the *DataAdapters*, you need to code each command properties just like the one example for the *Customers* table. On the Update method of the *DataSet*, you need to specify the table name:

```
Public Sub UpdateCustomerOrders(ByVal oDS As DataSet)


    'update the Customer DataTable of the DataSet
    Call UpdateCustomerDT(oDS)
    'update the Order DataTable of the DataSet
    Call UpdateOrdersDT(oDS)


End Sub
Private Sub UpdateCustomerDT(ByRef oDS As DataSet)
    Dim oConn As New SqlConnection("Password=;User ID=sa;" & _
        "Initial Catalog=Northwind;Data Source=(local)")
    'This creates a new command object
    Dim oCmdSelect As New SqlCommand()
    Dim oCmdInsert As New SqlCommand()
    Dim oCmdUpdate As New SqlCommand()
    Dim oCmdDelete As New SqlCommand()
    Dim oDA As SqlDataAdapter
    Dim oTran As SqlTransaction

    'Create the DataAdapter
    oDA = New SqlClient.SqlDataAdapter()
    With oDA
        .SelectCommand = oCmdSelect
        .InsertCommand = oCmdInsert
        .UpdateCommand = oCmdUpdate
        .DeleteCommand = oCmdDelete
    End With

        'configure command object for Selecting records
        'configure command object for Inserting records
        'configure command object for Updating records
```

```
            'configure command object for Deleting records

        oDA.Update(oDS, "Customers")

    End Sub
    Private Sub UpdateOrdersDT(ByRef oDS As DataSet)
        Dim oConn As New SqlConnection("Password=;User ID=sa;" & _
                     "Initial Catalog=Northwind;Data Source=(local)")
        'This creates a new command object
        Dim oCmdSelect As New SqlCommand()
        Dim oCmdInsert As New SqlCommand()
        Dim oCmdUpdate As New SqlCommand()
        Dim oCmdDelete As New SqlCommand()
        Dim oDA As SqlDataAdapter

        'Create the DataAdapter
        oDA = New SqlClient.SqlDataAdapter()
        With oDA
            .SelectCommand = oCmdSelect
            .InsertCommand = oCmdInsert
            .UpdateCommand = oCmdUpdate
            .DeleteCommand = oCmdDelete
        End With

            'configure command object for Selecting records
            'configure command object for Inserting records
            'configure command object for Updating records
            'configure command object for Deleting records

        oDA.Update(oDS, "Orders")

    End Sub
```

Say that you have a *DataSet* that has only a few changes made to it. Maybe you want to save network traffic by sending over only the modified or new records. It is not uncommon for an application to display a lot of records, and the user ends up modifying maybe one or two records. How do you send only a subset of the *DataSet*? This is accomplished by using the *GetChanges* method of the *DataSet* object or through the *DataTable* object of the *DataSet*.

When you call the *GetChanges* method on the *DataTable* object, the *DataTable* object returns a copy with only the changed records. When you call the *GetChanges* method on the *DataSet* object, the *DataSet* object returns a new *DataSet* with only the changed records. Whichever one you choose to use, you can specify what records you want. You can specify whether you want all the changed records or only the ones that are new, modified, or deleted.

Here's how to use *GetChanges* on the *DataSet*:

```
oChangedDS = oDS.GetChanges(DataRowState.Modified)
```

Here's how to use *GetChanges* on the *DataTable*:

```
oChangedDT = oDS.Tables(0).GetChanges(DataRowState.Modified)
```

Transactions

Because you are able to update multiple tables, what if you need to back out the database updates if something goes wrong in one of the tables? ADO.NET supports transactions. You can use the *Connection* object (*SQLConnection* and *OleDbConnection*) to perform transactional updates. On the *Connection* object, the *BeginTransaction* method returns a *Transaction* object. The *Transaction* object in turn is used to either *Commit* the changes, in which all changes made to the database are made permanent, or *Rollback* the changes, in which all changes to the database from the start of the transaction are dropped. Here is example code in using the transactional processing:

```
    Public Sub UpdateCustomerList(ByVal oDS As DataSet)
        Dim oConn As New SqlConnection("Password=;User ID=sa;" & _
                    "Initial Catalog=Northwind;Data Source=(local)")
        'This creates a new command object
        Dim oCmdSelect As New SqlCommand()
        Dim oCmdInsert As New SqlCommand()
        Dim oCmdUpdate As New SqlCommand()
        Dim oCmdDelete As New SqlCommand()
        Dim oDA As SqlDataAdapter
```

```
  Dim oTran As SqlTransaction

  oConn.Open()
  oTran = oConn.BeginTransaction()

  'Create the DataAdapter
  oDA = New SqlClient.SqlDataAdapter()
  With oDA
      .SelectCommand = oCmdSelect
      .InsertCommand = oCmdInsert
      .UpdateCommand = oCmdUpdate
      .DeleteCommand = oCmdDelete
  End With

With oCmdSelect
 .CommandText = "SELECT CustomerID, CompanyName, ContactName," & _
 " ContactTitle, Address, City, Region," & _
" PostalCode, Country, Phone, Fax FROM Customers"
 .Connection = oConn
  End With

  With oCmdInsert
      'configure command object for Inserting records
      .Transaction = oTran
  End With

  With oCmdUpdate
      'configure command object for Updating reocrds
      .Transaction = oTran
  End With

  With oCmdDelete
      'configure command object for Deleting records
      .Transaction = oTran
```

```
        End With

        Try
            oDA.Update(oDS)
            oTran.Commit()
        Catch oErr As Exception
            oTran.Rollback()
        End Try

        'Check on Connection Object, Close it
        If oConn.State = oConn.State.Open Then
            oConn.Close()
        End If

    End Sub
```

The code is very similar to code not using a transaction. What is different is that you got a *Transaction* object from the *Connection* object using the *BeginTransaction* method. In the *Command* objects that you want to enlist inside the transaction, you set the *transaction* property of those *Command* objects to the *Transaction* object. You then see if the updates are successful. If the updates are successful, you call the *Commit* method of the *Transaction* object to make the changes permanent. If the updates were not successful, you call the *Rollback* method of the *Transaction* object to drop all the changes that were made at the start of the transaction. At the end of the routine, you make sure that the database is closed.

If you ever want to know how records were affected, this is easy to do. All that is needed is to declare a variable to hold the count, and a slight change to calling the *Update* method of the *DataAdapter*. The *DataAdapter* can return a count of the number of records that were affected by the operations:

```
Dim iRowsUpdCnt As Integer
iRowsUpdCnt = oDA.Update(oDS, "Orders")
```

In some of the example code, notice that the database connection was explicitly opened and closed, whereas in other examples, there was no code for opening and closing the connection. The *DataAdapter* automatically takes care of opening and closing the connection. It is also smart enough to know the state of the connection. When you call the *Fill* method and the connection is closed, it will open it; once the *DataSet* is filled, the *DataAdapter* will close the connection. If the connection was already open when you call the *Fill* method, the *DataAdapter* will not close the connection. The *DataAdapter* will keep the connection open, and you would have to explicitly close the connection. What is the advantage of this behavior? The advantage is that if you need to extract data for more than one table, you get a performance gain by not opening and closing the connection every time you use the *Fill* method to extract data. You can open the connection once, and after you are done loading all your tables, you can then close it.

Also, when calling the *Update* method of the *DataAdapter*, the *Update* method essentially performs an *AcceptChanges* on the *DataSet*. What you must consider is how the *Update* method of the *DataAdapter* is called. What we mean is whether the *DataAdapter* is within your application or is within a component. If the *DataAdapter* is within your application, you would not necessarily need to call the *AcceptChanges* method of the *DataSet*. However, if the *DataAdapter* is within a component, you would have probably passed the *DataSet* to the component to be used by the *DataAdapter*'s *Update* method within it. It is more than likely that you passed the *DataSet* to the component as *ByVal*. In this case, once the component has completed its task of updating the data source via a *DataAdapter*, you would need to call the *AcceptChanges* method of the *DataSet*. Why is this necessary? Remember that passing an object as *ByVal* means that a copy of the object is used by the calling method and not the actual object. Modifications are made on the copy and not the actual object.

XML and ADO.NET

The World Wide Web Consortium (W3C) is the governing standards body for the Internet. It has approved Extensible Markup Language (XML), which has been widely accepted by many industries. With XML, you now have a way to easily exchange information between applications running on disparate platforms. All that you need is an XML parser. A *DataSet* can read data from XML documents. A *DataSet* can be written as an XML document.

You can fill a *DataSet* from an XML document. In the following code, once the schema and data is loaded into the *DataSet*, the code just iterates through the rows to load a list box with the values of "CustomerID":

```
Dim oDS As New DataSet()
Dim iCnt As Int32
```

```
        oDS.ReadXml("c:\CustomersData.xml", XmlReadMode.ReadSchema)
        If oDS.Tables.Count > 0 Then
            For iCnt = 0 To (oDS.Tables(0).Rows.Count - 1)
            ListBox1.Items.Add( _
            oDS.Tables(0).Rows(iCnt).Item("CustomerID").ToString)
            Next
        Else
            MessageBox.Show("No tables loaded")
        End If
```

You can create XML from a *DataSet* in order to exchange information. The following code simply connects to the Northwind database and retrieves the customer information to load a *DataSet* with the data and the schema. Once the *DataSet* is loaded, the *DataSet* (data and schema) is written as an XML file:

```
        Dim oConn As New SqlConnection("Password=a62b34gs;User ID=sa;" & _
                    "Initial Catalog=Northwind;Data Source=(local)")
        Dim oCmd As New SqlCommand("Select * from customers", oConn)
        Dim oDA As SqlDataAdapter
        Dim oDS As New DataSet()

        oDA = New SqlDataAdapter(oCmd)
        oDA.Fill(oDS, "Customers")
        oDA.FillSchema(oDS, SchemaType.Mapped, "Customers")
        oDS.WriteXml("c:\CustomersData.xml", XmlWriteMode.WriteSchema)
```

You may have noticed in the code example in the "Relationships and Constraints" section earlier in this chapter, the *Nested* property is set to True. The *Nested* property of the *DataRelation* has a default value of False. This property determines whether the child objects are nested within the parent objects. When calling the *WriteXML* method of the *DataSet*, the customer information and the order information will appear separate. When setting the *Nested* property to True, this will ensure that when you call the *WriteXML* method, the Order information will appear within each customer's information.

Here is what the XML would look like if the *Nested* property is set to False:

```
<CustomerOrders>
  <Customers>
    <CustomerID>ALFKI</CustomerID>
    <CompanyName>Alfreds Futterkiste</CompanyName>
  </Customers>
  <Orders>
    <OrderID>10643</OrderID>
    <CustomerID>ALFKI</CustomerID>
    <OrderDate>1997-08-25T00:00:00</OrderDate>
  </Orders>
</CustomerOrders>
```

Here is what the XML would look if the *Nested* property is set to True:

```
<CustomerOrders>
    <CustomerID>ALFKI</CustomerID>
    <Orders>
      <OrderID>10643</OrderID>
      <CustomerID>ALFKI</CustomerID>
      <OrderDate>1997-08-25T00:00:00</OrderDate>
    </Orders>
    <Orders>
      <OrderID>10692</OrderID>
      <CustomerID>ALFKI</CustomerID>
      <OrderDate>1997-10-03T00:00:00</OrderDate>
    </Orders>
    <CompanyName>Alfreds Futterkiste</CompanyName>
  </Customers>
</CustomerOrders>
```

The *DataSet* object is a relational representation of data. The Microsoft .NET Framework provides a way to represent the data in a hierarchical way. The .NET Framework lets you manipulate the data using the W3C Document Object Model (DOM). The DOM is an in-memory hierarchical representation of an XML document. You can navigate and edit the document using the DOM with the *XmlDataDocument* object. The *XmlDataDocument* object is an extension of the *XmlDocument* object that implements the DOM Level 1 Core and Level 2 Core interfaces. How the *XmlDataDocument* object extends the *XmlDocument* object is that it can expose the loaded XML document as a *DataSet*. Any changes made to the *DataSet* are reflected in the *XmlDataDocument*. Any changes in the

XmlDataDocument are reflected in the *DataSet*. You can mix XML and relational views of the underlying data. So, the *XmlDataDocument* acts as bridge between XML and the relational *DataSet*. Figure 6.7 is an example showing how you can synchronize an *XmlDataDocument* with a *DataSet*. This code is available at **www.syngress.com/solutions** in the Ch6WinApp project under the Form1.vb file.

Figure 6.7 ***XMLDataDocument*** **Example**

```
        Dim oConn As New SqlConnection("Password=a62b34gs;User ID=sa;" & _
                     "Initial Catalog=Northwind;Data Source=(local)")
        Dim oCmd As New SqlCommand("Select * from customers", oConn)
        Dim oDA As SqlDataAdapter
        Dim oDS As New DataSet()
        Dim xmlDoc As XmlDataDocument
        Dim oRootNode As XmlNode
        Dim oCustNode As XmlNode
        Dim oNode As XmlNode
        Dim iCnt As Int32
        Dim oDT As DataTable
        Dim oTempNode As XmlNode
        Dim oWriter As XmlTextWriter
        Dim xslTran As XslTransform

        'load up the dataset with data and schema
        oDA = New SqlDataAdapter(oCmd)
        'set the select command to fill dataset
        oDA.Fill(oDS, "Customers")
        oDA.FillSchema(oDS, SchemaType.Mapped, "Customers")

        'synchronize XmlDataDocument with the dataset
        xmlDoc = New XmlDataDocument(oDS)

        'look at what in dataset exposed by XmlDataDocument
        With xmlDoc.DataSet.Tables(0)
            Label1.Text = Label1.Text & .Rows.Count.ToString
```

Continued

Figure 6.7 Continued

```
          For iCnt = 0 To (.Rows.Count - 1)
               'display customer id in list box
               ListBox1.Items.Add(.Rows(iCnt).Item(0).ToString)
          Next
     End With

     'point to the root node
     oRootNode = xmlDoc.FirstChild

     xmlDoc.DataSet.EnforceConstraints = False

     'remove a node
     oTempNode = oRootNode.FirstChild
     oRootNode.RemoveChild(oTempNode)

     'display rows using dataset
     'get the "Customers" DataTable
     With xmlDoc.DataSet
          oDT = .Tables(0).GetChanges(DataRowState.Unchanged)
     End With

     With oDT
          Label2.Text = Label2.Text & .Rows.Count.ToString
          For iCnt = 0 To (.Rows.Count - 1)
               'display customer id in list box
               ListBox2.Items.Add(.Rows(iCnt).Item(0).ToString)
          Next
     End With

     'reject the changes
     xmlDoc.DataSet.RejectChanges()

     'display rows using XmlDataDocument
     Label3.Text = Label3.Text & _
```

Continued

Figure 6.7 Continued

```
                  xmlDoc.DataSet.Tables(0).Rows.Count.ToString
        For Each oCustNode In oRootNode.ChildNodes
            For Each oNode In oCustNode.ChildNodes
                If oNode.Name = "CustomerID" Then
                    ListBox3.Items.Add(oNode.InnerText.ToString)
                    Exit For
                End If
            Next
        Next

        'use xsl to create a new xml
        xslTran = New XslTransform()
        xslTran.Load("c:\CustomersTransform.xsl")
        oWriter = New XmlTextWriter("c:\xslt_output.xml", _
                                    System.Text.Encoding.UTF8)
        oWriter.WriteStartDocument()
        xslTran.Transform(xmlDoc, Nothing, oWriter)
        oWriter.WriteEndDocument()
        oWriter.Close()
```

Just as before, make the connection to the data source through the *DataAdapter* and use the *Fill* method to retrieve and store the rows in a *DataTable*, *Customers*. Use the *FillSchema* method to retrieve the structure information (columns information, constraints, and so on). Then create a new *XmlDataDocument* object, giving it the *DataSet* object containing my information. That's all that you need to do to synchronize the *XmlDataDocument* with a *DataSet*. Now, you can either do updates on the *XmlDataDocument* or on the *DataSet* exposed by the *XmlDataDocument* and have the changes reflected in both. In the code example, first fill a list box with the current rows to show the available rows in the *DataSet*. Then use the *XmlDataDocument* to drill down to one of the nodes, a customer row. Use the *RemoveNode* method of the *XmlDataDocument* to delete one of the *Customer* rows. To show that the delete is reflected in the *DataSet*, spin through the rows of the exposed *DataSet* to fill up a second list box. You'll see that when comparing the first list box with the second list box, you'll notice that there is one less row. Now, to show that what you do in the *DataSet* is reflected in the *XmlDataDocument*, I invoke the *RejectChanges* method on the *DataSet*. You see that when the third list box is filled, the original row that was deleted is back again in the list.

The last part of the code example uses the *XslTransform* object. The *XmlDataDocument* allows you to use the features of Extensible Stylesheet Language (XSL) to write the *DataSet* into another format. Here is the XSL file that I used to transform the *XmlDataDocument*. In the XSL document, I am creating an XML document that has only the CustomerIDs. Here is the XSL file that I used for the transformation:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
<xsl:template match="/">
    <CustomerList>
        <xsl:for-each select="//Customers">
            <Customer>
                <ID><xsl:value-of select="CustomerID"/></ID>
            </Customer>
        </xsl:for-each>
    </CustomerList>
</xsl:template>
</xsl:stylesheet>
```

Here is sample output of the transform:

```
<?xml version="1.0" encoding="utf-8"?>
      <CustomerList>
               <Customer>
                     <ID>ALFKI</ID>
               </Customer>
               <Customer>
                     <ID>ANATR</ID>
               </Customer>
               <Customer>
                     <ID>ANTON</ID>
               </Customer>
```

```
                    <Customer>
                          <ID>AROUT</ID>
                    </Customer>
                    <Customer>
                    .
                    .
                    .
          <CustomerList>
```

So, if you have an application that requires an XML document in a different format than your *DataSet*, you can use XSL transformation. In the *System.Xml* namespace, you can see the XML features supported by the .NET Framework. The Framework offers many more features than what you have seen in these examples.

Summary

ADO.NET is not "just another Data Access Model" from Microsoft. In order to support Microsoft's new technology, Microsoft .NET Framework, and its vision, a new way of accessing data was needed. ADO.NET is a new Data Access model built from the ground up. ADO.NET strongly supports the .NET Framework by the way it knows XML, how it avoids the restrictions of the COM world, and how it leverages XML for interoperability. ADO.NET was built with scalability in mind.

ADO.NET goes beyond ADO. It has a flexible and powerful architecture. ADO.NET has two objects, *DataReader* and *DataSet*, that you can use to access data. The *DataReader* is a noncached, high-performance data access object to retrieve information. You cannot use the *DataReader* to directly update the data source or traverse the information. The *DataSet* is an in-memory cache of data that can contain multiple tables. The tables in the *DataSet* can have relations and constraints to ensure referential integrity as well as data integrity on the client-side. With the *DataAdapter*, you now have full control of retrieving and updating data from the data source.

The *DataSet* can be persisted and transmitted over the wire as XML. You can also synchronize the *DataSet* with an *XmlDataDocument*. The *XmlDataDocument* is an extension of the *XmlDocument* object. The *DataSet* is a relational view data. XML is a hierarchical view of data. The *XmlDataDocument* acts as a bridge between XML and the *DataSet*. Once you have synchronized the *XmlDataDocument* with a *DataSet*, any changes that are made in the *XmlDataDocument* is automatically reflected in the *DataSet* and vice versa. If you need to transform the relational data in the *DataSet* into another format, you can synchronize the *DataSet* with an *XmlDataDocument* and use the *XmlDataDocument* as input to an XSLT transform object.

So, where do you go from here? Many more of the powerful features in ADO.NET weren't covered in this chapter. Go and try it out. Start coding.

Solutions Fast Track

Introduction to ADO.NET

☑ We had previous Data Access Models (DAO, ODBC, RDO, ADO, and OleDB).

☑ Microsoft re-examines how businesses consume data.

☑ Microsoft .NET Framework is a new technology that you can use to provide business solutions at the Enterprise level and also between business entities.

Why ADO.NET?

☑ Microsoft didn't just slap on the .NET extension. ADO.NET was built from the ground up.

☑ ADO.NET strongly supports the .NET Framework and the .NET Framework's vision of interoperability.

☑ ADO.NET was designed with scalability in mind.

ADO.NET Object Model

☑ Anyone that has been using ADO should see familiarities with ADO.NET.

☑ ADO.NET has two .NET Data Providers. Microsoft SQL Server .NET Data Provider is optimized for Microsoft SQL Server. OLE DB .NET Data Provider is for data exposed through OleDB.

☑ You can use two objects, *DataReader* and *DataSet*, to retrieve data.

Using the *DataReader*

☑ *DataReader* is read-only and forward-only.

☑ *DataReader* is a high-performance data access object.

☑ *DataReader* is noncached data, which means once you move off the record, it is no longer available.

Using the *DataSet*

☑ As opposed to the *DataReader*, *DataSet* is an in-memory cache of data.

☑ The *DataSet* uses the *DataReader* underneath to retrieve the data.

☑ The *DataSet* is disconnected.

☑ The *DataAdpater* and the *DataSet* work hand-in-hand. The *DataAdpater* is the object that lets you exchange data between the data source and the a *DataSet*.

☑ The *DataSet* can contain one or more *DataTable*s, a table of in-memory cache data.

☑ The *DataSet* has a collection of *DataRelation*s that are relations between the *DataTable*s.

☑ The *DataTable* contains *DataRows*, *DataColumns*, and *Constraints*.

☑ Unlike ADO, you can have control of the batch updates.

XML and ADO.NET

☑ The W3C has approved it as an industry standard.

☑ A *DataSet* can read data from XML documents.

☑ The *DataSet* can be written as an XML document.

☑ You can synchronize a *DataSet* with an *XmlDataDocument* and use XSLT transformation to transform the *DataSet* into a different format.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: Do you recommend some of the naming conventions that were in the examples?

A: No. Some of the names that we used for variables were shortened so that the code could easily fit on one line of text without wrapping to another line.

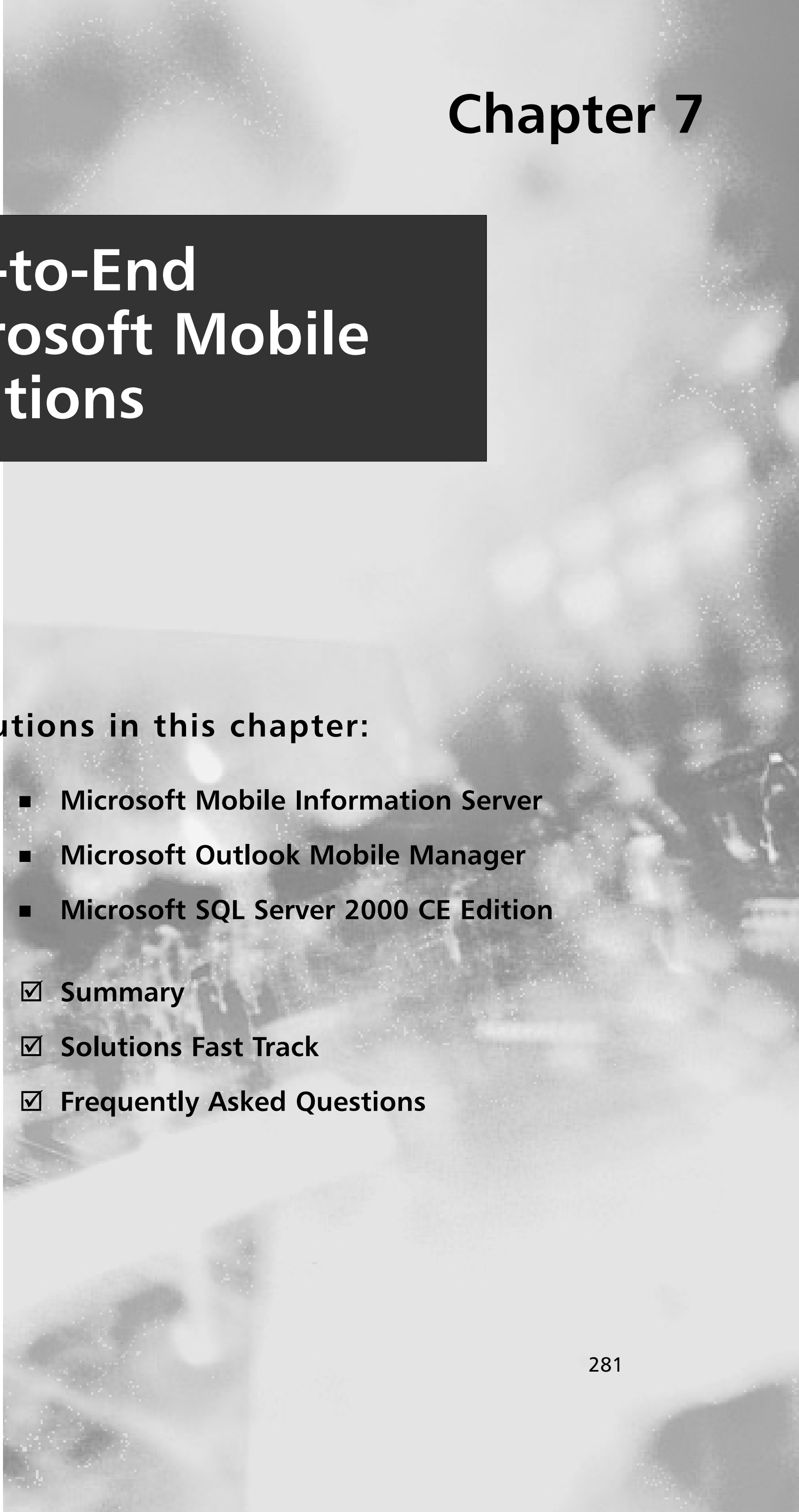
Q: So, we have seen the *DataReader* object and the *DataSet* object. When would you use one over the other?

A: In most situations, the *DataReader* object is the choice of data access in developing ASP.NET applications because of its high performance. Use the *DataSet* in the following instances:

- When you need to do batch updates
- When you need more than one *DataTable*
- When you need to take advantage of synchronizing your data with an *XmlDataDocument*
- When you need to move forward and backwards of the rows

Q: Do any wizards come with Visual Studios.NET that would help with the *DataSet*?

A: Yes. A Data Configuration Wizard automatically pops up when you drop a *DataAdapter* onto a form. It will help you configure the *DataAdapter* and the commands. It can also create the stored procedures for you.



Chapter 7

End-to-End Microsoft Mobile Solutions

Solutions in this chapter:

- Microsoft Mobile Information Server
- Microsoft Outlook Mobile Manager
- Microsoft SQL Server 2000 CE Edition

☑ Summary

☑ Solutions Fast Track

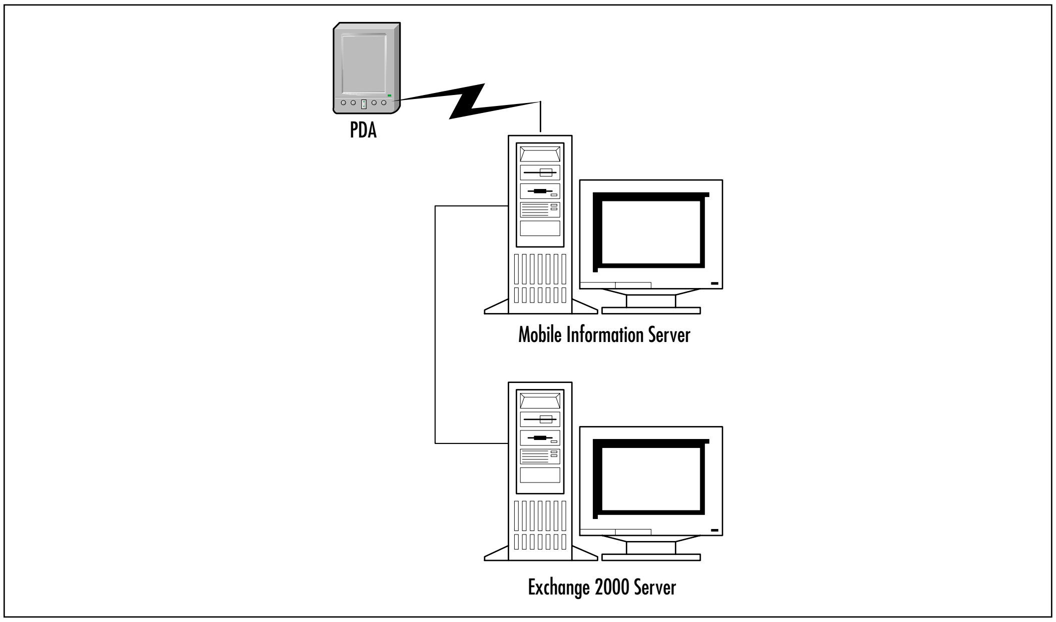
☑ Frequently Asked Questions

Introduction

This chapter takes you from start to finish with a mobile solution using key Microsoft technologies. The goal is to make vital desktop services available to users who are in and out of the office. The solutions here include installing, configuring, and generally troubleshooting the infrastructure that lets mobile workers connect to your intranet and collect their messages and appointments on a mobile phone or Pocket PC. You'll also see an application that allows the same users to take selected corporate data with them on their Pocket PCs for use on the road.

Before deploying an enterprise application using these technologies, you need to bring yourself up to speed and work out the kinks in a lab environment. You don't want to risk the expense, stress, and wrath of colleagues by causing downtime on a mission-critical server while learning to use leading-edge software. Keep in mind that "backing out" some of the settings due to an error can be very time consuming. Therefore, the solutions presented here show the installation and configuration in an isolated Windows domain using two lab computers and a Pocket PC that are configured as shown in Figure 7.1. Once you are comfortable that the system works as expected in the protected setting, you'll be ready to build it again in a production environment.

Figure 7.1 LAN Setup for Mobile Solution



We explore the requirements of each technology, the installation, and the configuration. Then we'll build its portion of the overall solution. By the end of this chapter, you'll have a small demonstration project that you can customize and expand. The software you need for testing is available from Microsoft in demonstration or trial versions that are freely downloadable. Therefore, you can "kick the tires" and develop a proof of concept project that costs no more than an investment of your time. This chapter uses the versions that ship with Microsoft's MSDN Universal subscription service and therefore may have developer features not in the regular product.

The project starts with Mobile Information Server (MIS) and Exchange 2000. After that, we hook in Mobile Outlook, the Mobile Outlook Manager, and at the end, a practical SQL Server 2000 CE Edition solution for the Pocket PC.

Microsoft Mobile Information Server

Microsoft Mobile Information Server (MIS) 2001 helps you tie together mobile applications and give your Wireless Application Protocol (WAP) and Pocket PC users access to services on your intranet. By tying MIS with Exchange Server 2000, you open a rich array of desktop services to these tiny portable devices. As you'll see, MIS acts as a "go-between" or proxy that relays messages back and forth between the corporate intranet and the mobile device.

Versions of MIS

There are two versions of Mobile Information Server 2001: Enterprise and Carrier. The Enterprise edition is the basic product for corporate use. It sits behind the corporate firewall and allows browsing and delivery of company data. Microsoft estimates that the Enterprise edition can handle a million notification messages a day; the Carrier edition can handle three million messages. The Carrier edition is hosted at a wireless operator's data center and provides an access point that serves numerous corporations. You won't likely encounter the Carrier version unless you work for a telecommunications carrier or Internet service provider that provides mobile data services to business and the general public. This chapter discusses only the Enterprise edition.

Preparing for MIS

This section ensures that you have what you need to evaluate and use MIS in your two-PC intranet lab. For ease of understanding, we call the machines Exch (the computer running Exchange 2000) and MIS (running Mobile Information

Server). The requirements are prerequisites to installing MIS. Don't install MIS yet—we walk through the procedures.

- **Exch Computer Prerequisites:**
  - Windows 2000 Server with service pack 1 or higher.
  - Active Directory with this machine configured as the domain controller.
  - Exchange 2000 Server with Exchange service pack 1. This includes the SMTP server.
  - Internet Information Server with the Message Queuing Server option.
- **MIS Computer Prerequisites:**
  - Windows 2000 Server with service pack 1 or higher.
  - Internet Information Server with the Message Queuing Server and SMTP server options.
  - This machine must *not* be running Exchange Server.

Prepare the basic domain before moving to the next step. Because this is a proof-of-concept installation, we don't worry about meeting the hardware requirements of a heavy-duty enterprise application. We assume that your machines are adequate for hosting the software and accommodating a few test users.

Installing Mobile Information Server

MIS relies on the Active Directory and Internet Information Server. Therefore, the installation takes place on both Exch (which is the name of our domain controller) and on the machine named MIS. In all cases, log on as the domain administrator, not just as the local machine administrator.

Updating the Active Directory Schema

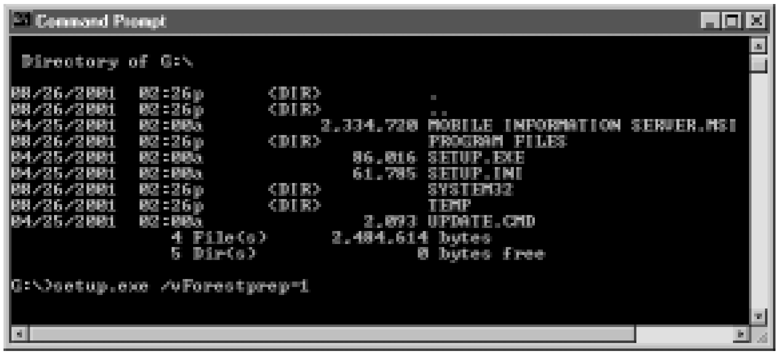
The installation begins by updating and extending the Active Directory schema for use by MIS. You need to log on to the Exch machine as the domain administrator:

1. Insert the **MIS CD** into the Exch machine but do not let the installation routine start. Cancel it if it starts automatically.
2. Open a command window and navigate to the Setup directory for MIS.

3. As shown in Figure 7.2, execute the following command:

   ```
   setup /vForestprep=1
   ```

Figure 7.2 Command Line Instruction to Extend the Active Directory

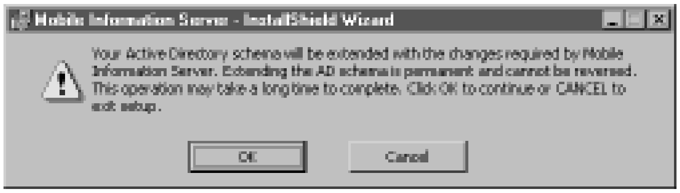


Command Prompt

```
 Directory of G:\

08/26/2001  02:26p      <DIR>          .
08/26/2001  02:26p      <DIR>          ..
04/25/2001  02:00a          2.334.720 MOBILE INFORMATION SERVER.MSI
08/26/2001  02:26p      <DIR>          PROGRAM FILES
04/25/2001  02:00a             86.016 SETUP.EXE
04/25/2001  02:00a             61.785 SETUP.INI
08/26/2001  02:26p      <DIR>          SYSTEM32
08/26/2001  02:26p      <DIR>          TEMP
04/25/2001  02:00a              2.093 UPDATE.CMD
               4 File(s)      2.484.614 bytes
               5 Dir(s)               0 bytes free

G:\>setup.exe /vForestprep=1
```

4. When prompted for confirmation, as shown in Figure 7.3, click **OK**.

Figure 7.3 MIS Extends the Active Directory Schema and Adds a Microsoft Mobility Admins Group

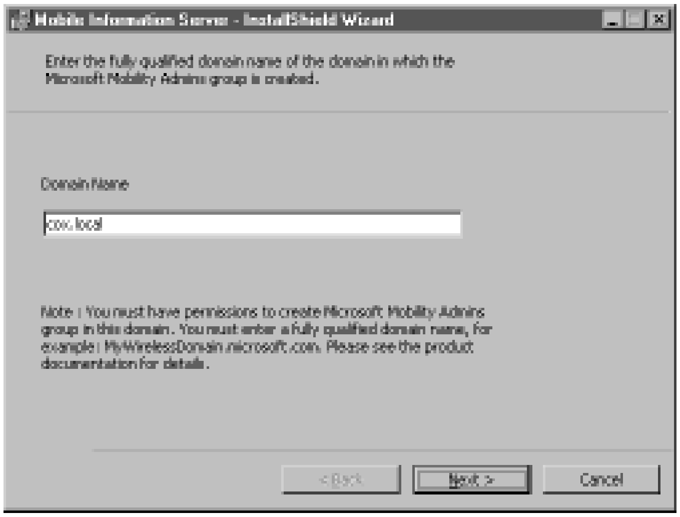


Mobile Information Server - InstallShield Wizard

Your Active Directory schema will be extended with the changes required by Mobile Information Server. Extending the AD schema is permanent and cannot be reversed. This operation may take a long time to complete. Click OK to continue or CANCEL to exit setup.

5. As shown in Figure 7.4, enter the name of your domain and click **Next**. This example is running in a lab that uses COX.LOCAL as the domain name. Your domain name may be MYCORP.COM. It is important that this is the domain to which Exch and MIS belong.
6. Note that extending and populating the Active Directory to accommodate Mobile Information Server takes a few minutes. When the wizard completes the activity, as shown in Figure 7.5, click **Finish**.
7. Open Active Directory Users and Computers and confirm that the setup has added a new administration group called Microsoft Mobility Admins as shown in Figure 7.6.

Figure 7.4 To Extend the Active Directory, the Installer Requires the Domain Name



Mobile Information Server - InstallShield Wizard

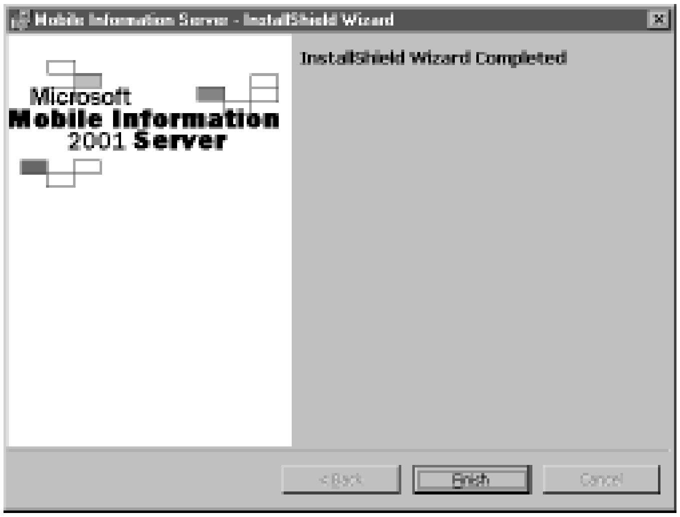
Enter the fully qualified domain name of the domain in which the Microsoft Mobility Admins group is created.

Domain Name

cox.local

Note : You must have permissions to create Microsoft Mobility Admins group in this domain. You must enter a fully qualified domain name, for example : MyWirelessDomain.microsoft.com. Please see the product documentation for details.

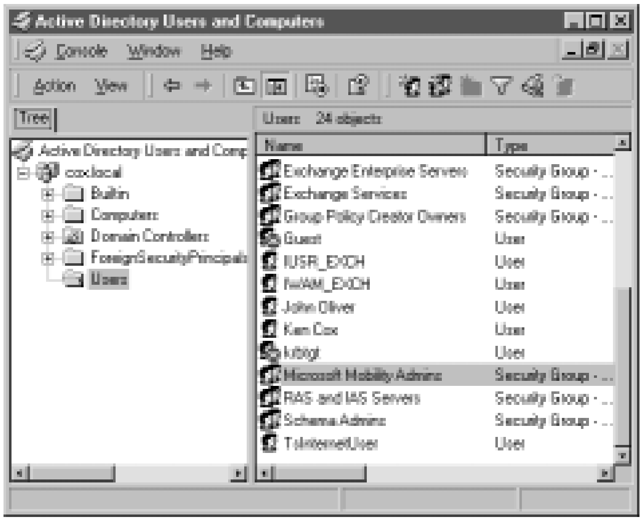
Figure 7.5 After Several Minutes, the Wizard Finishes Extending the Active Directory



Mobile Information Server - InstallShield Wizard

InstallShield Wizard Completed

Figure 7.6 Active Directory Shows the Addition of a New Administrative Group

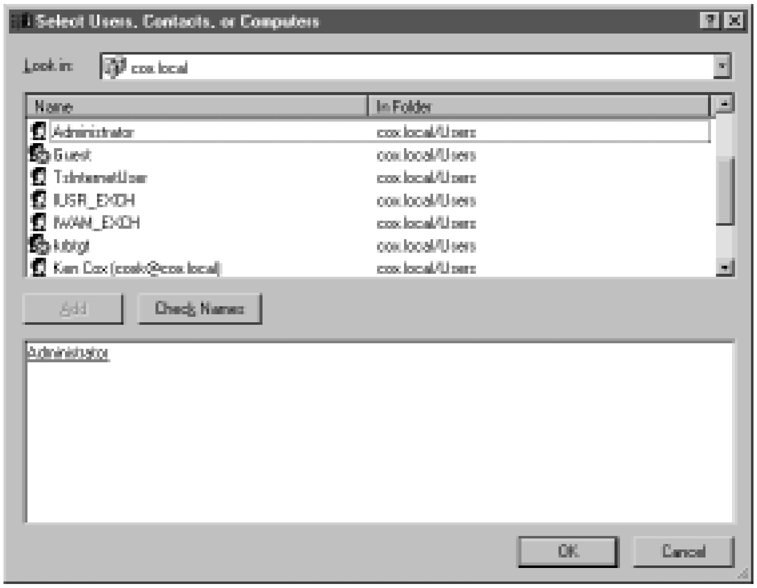

Active Directory Users and Computers
Console Window Help
Action View
Tree
Active Directory Users and Comp
cox.local
Builtin
Computers
Domain Controllers
ForeignSecurityPrincipals
Users

Users 24 objects

Name	Type
Exchange Enterprise Servers	Security Group - ...
Exchange Services	Security Group - ...
Group Policy Creator Owners	Security Group - ...
Guest	User
IUSR_EXCH	User
IWAM_EXCH	User
John Oliver	User
Ken Cox	User
krbtgt	User
Microsoft Mobility Admins	Security Group - ...
RAS and IAS Servers	Security Group - ...
Schema Admins	Security Group - ...
TsInternetUser	User

8. Add the domain administrator to the Microsoft Mobility Admins group. To do this, double-click the group's name and on the **Members** tab, click **Add**. Figure 7.7 shows the addition of the Administrator.

Figure 7.7 Add the Domain Administrator to the Microsoft Mobility Admins Group



Name	In Folder
Administrator	cox.local/Users
Guest	cox.local/Users
TsInternetUser	cox.local/Users
IUSR_EXCH	cox.local/Users
IWAM_EXCH	cox.local/Users
krbtgt	cox.local/Users
Ken Cox (coxk@cox.local)	cox.local/Users

Administrator

9. Close **Active Directory Users and Computers**. This completes the first stage of the Exch machine preparation.

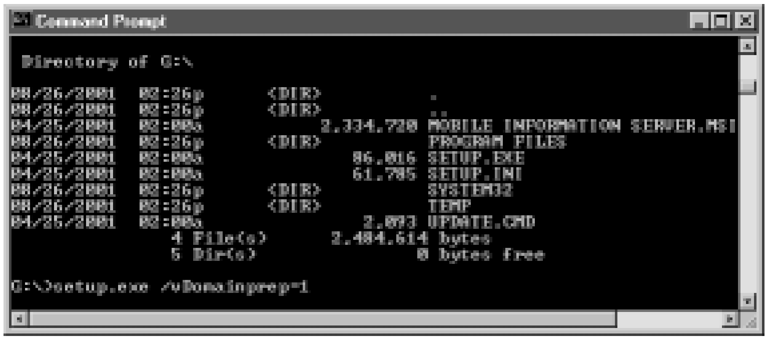
Preparing the Mobile Information Server Accounts

After extending the Active Directory schema, you still need to prepare the domain and add accounts for use by MIS. While still logged on to the Exch machine as a domain administrator, carry out the following steps:

1. As you did previously, open a command window and execute the following command, as shown in Figure 7.8:

   ```
   setup /vDomainprep=1
   ```

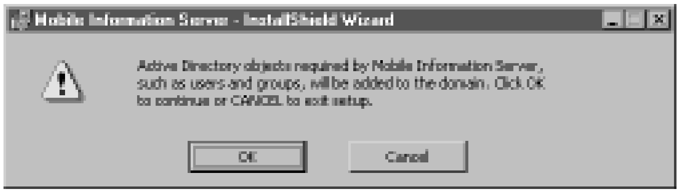
   Figure 7.8 Preparing the Exchange Domain Uses the Command Line

   ```
    Directory of G:\

   08/26/2001  02:26p      <DIR>          .
   08/26/2001  02:26p      <DIR>          ..
   04/25/2001  02:00a          2,334,720 MOBILE INFORMATION SERVER.MSI
   08/26/2001  02:26p      <DIR>          PROGRAM FILES
   04/25/2001  02:00a             86,016 SETUP.EXE
   04/25/2001  02:00a             61,785 SETUP.INI
   08/26/2001  02:26p      <DIR>          SYSTEM32
   08/26/2001  02:26p      <DIR>          TEMP
   04/25/2001  02:00a              2,093 UPDATE.CMD
                 4 File(s)      2,484,614 bytes
                 5 Dir(s)               0 bytes free

   G:\>setup.exe /vDomainprep=1
   ```

2. When prompted, as shown in Figure 7.9, confirm that you want to add users and group accounts to the domain and click **OK**.

   Figure 7.9 Confirm That You Want to Add MIS Users and Groups to the Domain

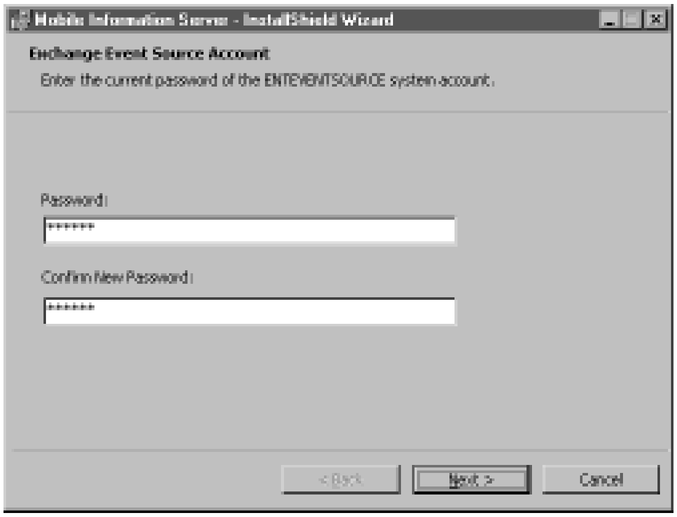
   

   Active Directory objects required by Mobile Information Server, such as users and groups, will be added to the domain. Click OK to continue or CANCEL to exit setup.

3. When prompted each time, provide (and record for later use) a password for each of the MIS system accounts as shown in Figure 7.10. (These accounts are Exchange Event Source, Message Processor, and HTTP Connector.) Click **Next** each time. The setup takes a few minutes to

add the accounts. This completes the first phase of the MIS installation on the Exch machine.

Figure 7.10 The Addition of MIS System Accounts Requires Three Passwords


Mobile Information Server - InstallShield Wizard
Exchange Event Source Account
Password:
Confirm New Password:
< Back
Next >
Cancel

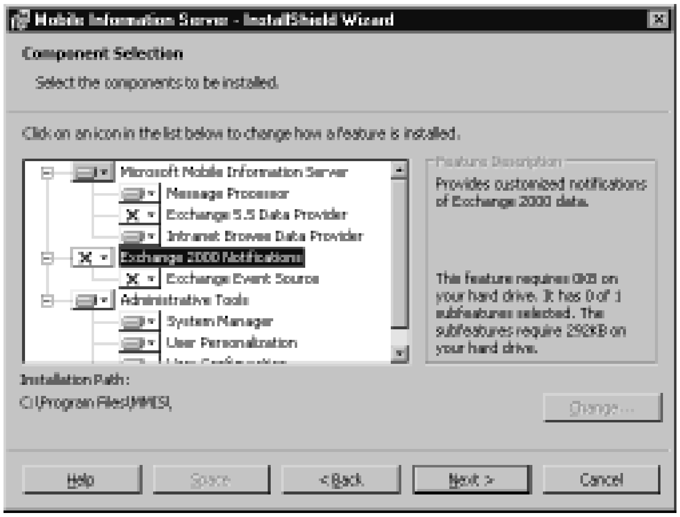
Enter the current password of the ENTEVENTSOURCE system account.

Installing the MIS Software on Its Own Server

At this point you have used two MIS utilities to configure the Active Directory and add service accounts to the domain. The actual installation of MIS is done on the dedicated MIS server. As the domain administrator, log on to the MIS machine and perform the setup steps:

1. Insert the MIS CD into the MIS machine and run **Setup.exe** without any command-line switches.
2. On the Welcome window, click **Next**.
3. On the License Agreement window, accept the agreement and then click **Next**. Confirm that you have read the license and then click **Next**.
4. If prompted for a CD key, enter the key number.
5. On the Wireless Accounts window, select **Use existing accounts** and then click **Next**.
6. On the Component Selection window, select all of the components except the Exchange 5.5 connector and Exchange Events (see Figure 7.11) and then click **Next**.

Figure 7.11 Deselect Exchange 2000 Notifications in the Component Selection Window



Component Selection

Select the components to be installed.

Click on an icon in the list below to change how a feature is installed.

- Microsoft Mobile Information Server
  - Message Processor
  - Exchange 5.5 Data Provider
  - Intranet Browse Data Provider
- Exchange 2000 Notifications
  - Exchange Event Source
- Administrative Tools
  - System Manager
  - User Personalization

Feature Description

Provides customized notifications of Exchange 2000 data.

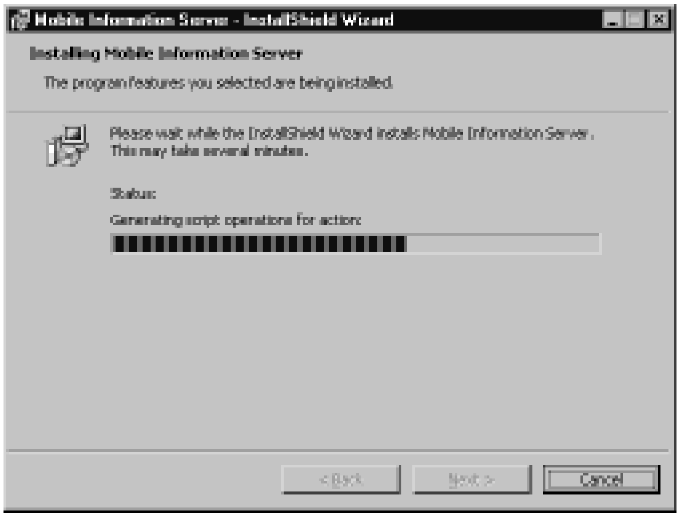
This feature requires 0KB on your hard drive. It has 0 of 1 subfeatures selected. The subfeatures require 292KB on your hard drive.

Installation Path:

C:\Program Files\MMIS\

7. When prompted for the Message Processor and HTTP Connector accounts, type their respective passwords that you used in the previous section, "Preparing the Mobile Information Server Accounts."
8. Step through the remaining confirmations. The setup routine begins the installation as shown in Figure 7.12.

Figure 7.12 The Mobile Information Server Setup Takes a Few Minutes to Complete



Installing Mobile Information Server

The program features you selected are being installed.

Please wait while the InstallShield Wizard installs Mobile Information Server. This may take several minutes.

Status:

Generating script operations for action:

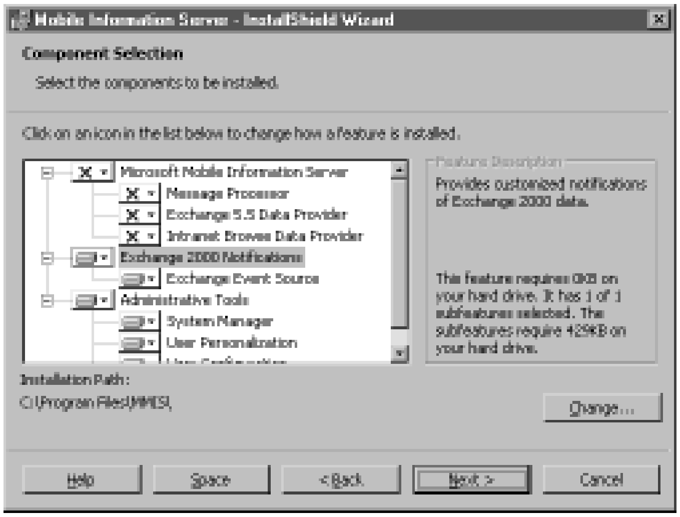
9. When the setup is finished, log off the MIS machine. This completes the software installation on the MIS machine.

Installing the Exchange 2000 Notification Components on the Exch Machine

Now that the system accounts are in place and MIS has been installed, you can install the Exchange 2000 Notification components on the Exch machine. Do not install the complete MIS product on the Exch machine. Be sure that you are logged on as the domain administrator.

1. Insert the MIS CD into the Exch machine and run **Setup.exe** without any command-line switches.
2. Step through the Welcome window and License Agreement screens and when prompted, enter your product identification number and click **Next**.
3. On the Component Selection window, clear the **Microsoft Mobile Information Server** components and select the **Exchange 2000 Notifications** as shown in Figure 7.13. Click **Next**.

Figure 7.13 Install the Exchange 2000 Notifications Components on the Exch Machine



Mobile Information Server - InstallShield Wizard

Component Selection

Select the components to be installed.

Click on an icon in the list below to change how a feature is installed.

- Microsoft Mobile Information Server
  - Message Processor
  - Exchange 5.5 Data Provider
  - Intranet Browse Data Provider
- Exchange 2000 Notifications
  - Exchange Event Source
- Administrative Tools
  - System Manager
  - User Personalization

Feature Description

Provides customized notifications of Exchange 2000 data.

This feature requires 0KB on your hard drive. It has 1 of 1 subfeatures selected. The subfeatures require 429KB on your hard drive.

Installation Path:

C:\Program Files\MMIS\

4. Type the password for the ENTEVENTSOURCE system account and then click through the remaining screens. The setup wizard installs the components.
5. When the installation is complete, click **Finish**.

WARNING

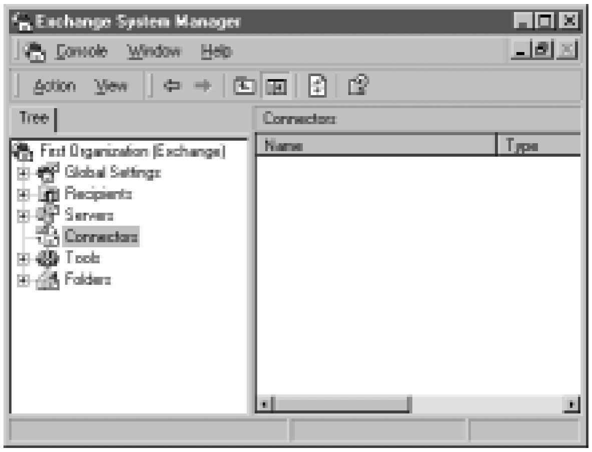
Do not attempt to install Mobile Information Server on a computer that is running Exchange Server. They are not compatible.

Configuring Exchange 2000 to Send Notifications to Mobile Information Server

Connectors are used as paths to send messages from one system or component to another. Your Exchange 2000 server needs to send notification messages to Mobile Information Server through a connector. Follow these steps to add a connector:

1. On the Exch machine, open the Exchange System Manager and select the **Connectors** node, as shown in Figure 7.14.

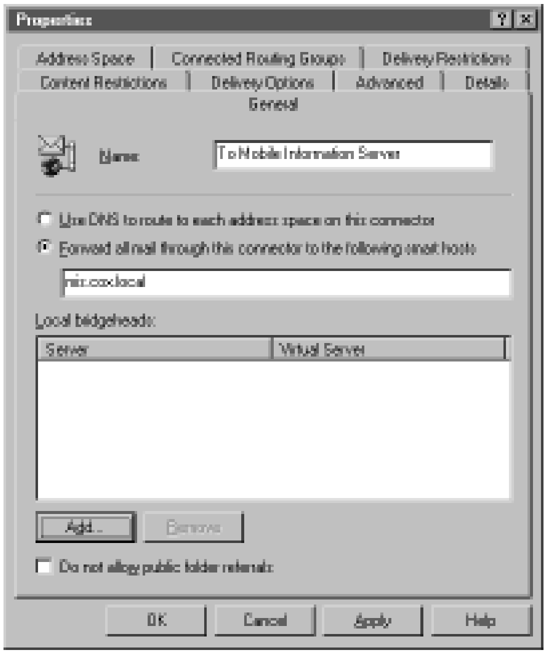
Figure 7.14 Adding a Connector in Exchange System Manager



2. Right-click the **Connectors** node, from the context menu select **New | SMTP Connector**.

3. As shown in Figure 7.15, name the connector **To Mobile Information Server** and enter the complete DNS address of the MIS machine.

Figure 7.15 Configuring the Name and Routing of the SMTP Connector



Name: To Mobile Information Server

( ) Use DNS to route to each address space on this connector

(•) Forward all mail through this connector to the following smart hosts

mis.cox.local

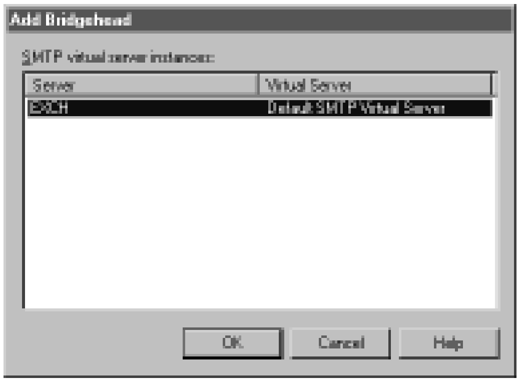
Local bridgeheads:

Server	Virtual Server

[ ] Do not allow public folder referrals

4. Click **Add**. In the Add Bridgehead window, select your server's name and click **OK**. See Figure 7.16.

Figure 7.16 Add Your Exchange Server as a Bridgehead

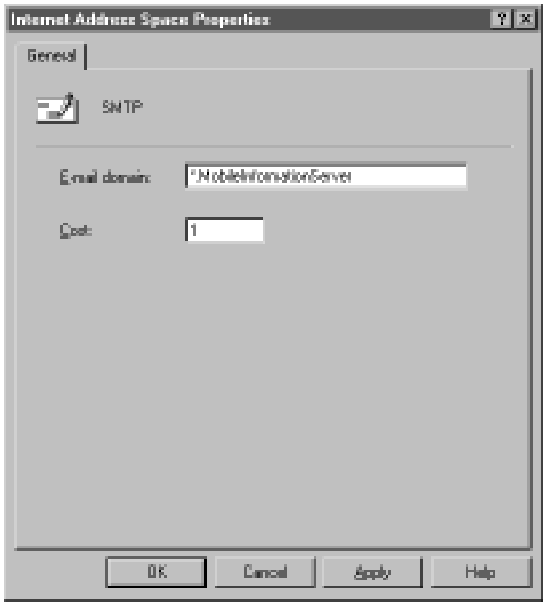


SMTP virtual server instances:

Server	Virtual Server
EXCH	Default SMTP Virtual Server

5. On the Address Space tab, click **Add** then select **SMTP** and click **OK**.
6. In the E-mail domain box, type ***.MobileInformationServer** as shown in Figure 7.17 and then click **OK**.

Figure 7.17 Provide the an E-Mail Domain Name Called *.MobileInformationServer


Internet Address Space Properties

General

SMTP

E-mail domain: *.MobileInformationServer

Cost: 1

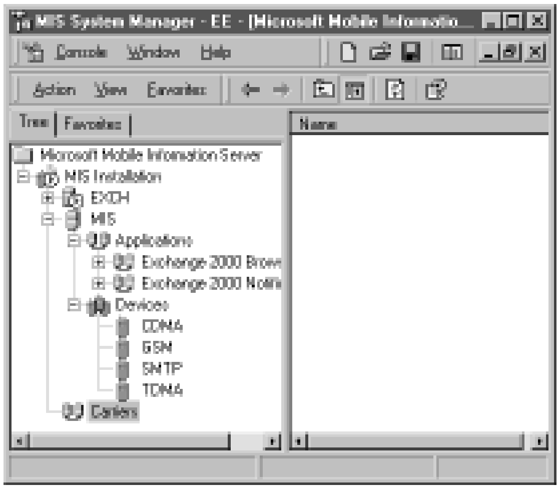
7. Click **OK** to close the properties page and close Exchange System Manager. This completes the configuration of an Exchange 2000 SMTP connector.

Configuring the SMTP Carrier to Send Push Notifications

Exchange 2000 can send short notifications to devices about the arrival of e-mail or other events. Users of mobile devices can configure what types of messages they want to receive as push notifications. For example, they can designate messages from certain senders to be forwarded through a push notification. To implement push notifications, you need to configure an SMTP carrier in MIS System Manager as follows:

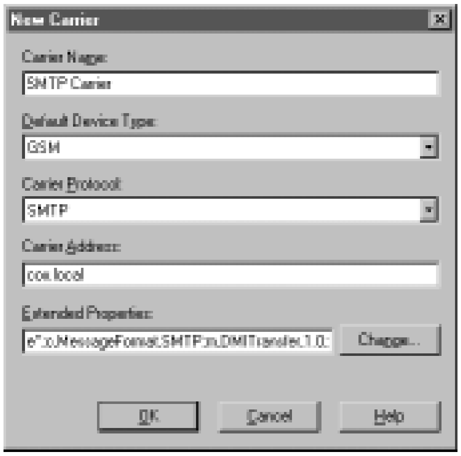
1. On the Exch machine, open the MIS System Manager and expand the MIS server node to reveal the Carriers node, as shown near the bottom of Figure 7.18.

Figure 7.18 Using MIS System Manager to Add a Carrier



2. Right-click the **Carriers** node and from the context menu, click **New Carrier**.
3. In the New Carrier dialog box, type or select the following values and then click **OK** (see Figure 7.19):
   - Carrier Name: **SMTP Carrier**
   - Default Device Type: **GSM**
   - Carrier Protocol: **SMTP**
   - Carrier Address: Use the name of the SMTP carrier in your lab environment, for example **mylab.com**.
   - Extended Properties: Leave the default values.
4. Notice that the SMTP Carrier has been added to the Carriers node in the MIS System Manager. This completes the configuration of the SMTP Carrier.

Figure 7.19 Adding a New SMTP Carrier



Carrier Name: SMTP Carrier

Default Device Type: GSM

Carrier Protocol: SMTP

Carrier Address: cox.local

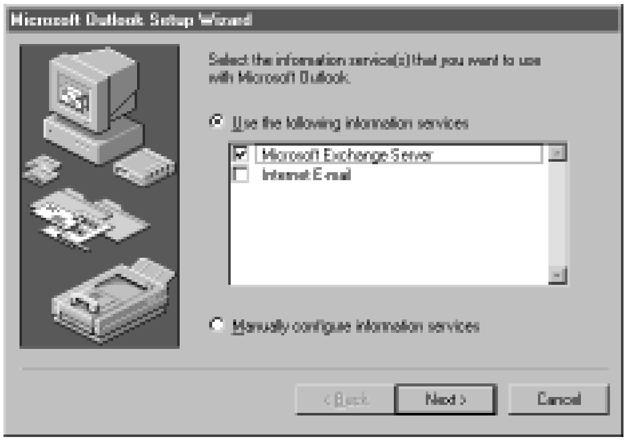
Extended Properties: e":o;MessageFormat:SMTP;n;DMITransfer;1.0:

Creating a Test Mailbox for the Administrator

Before you can access a mailbox from a mobile device, you need to initialize the Exchange 2000 mailbox. Without the initialization steps, the Outlook Mobile Access page will not appear or parts of the page will not work. In this procedure, you install and configure the desktop version of Microsoft Outlook on the Exch machine for use by the administrator:

1. While logged in as the domain administrator on the Exch machine, install the desktop version of Microsoft Outlook. Use the default settings for the administrator account.
2. After completing the Outlook installation, go to the Windows Control Panel and open the Mail applet.
3. On the General tab, click **Add**.
4. In the dialog box, select **Microsoft Exchange Server**, as shown in Figure 7.20, and then click **Next**.
5. Enter the name of the Microsoft Exchange server and domain name. In this lab setup, as shown in Figure 7.21, the server name is **Exch** and the domain is **cox.local**. Enter **Administrator** as the mailbox name and click **Next**.

Figure 7.20 Add the Microsoft Exchange Service to Mail Profile



Microsoft Outlook Setup Wizard

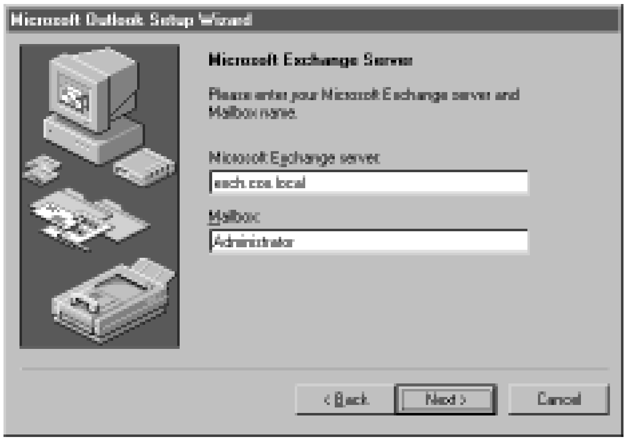
Select the information service(s) that you want to use with Microsoft Outlook.

(●) Use the following information services

[✓] Microsoft Exchange Server
[ ] Internet E-mail

( ) Manually configure information services

Figure 7.21 Enter the Exchange Server Name and Mailbox Name



Microsoft Exchange Server

Please enter your Microsoft Exchange server and Mailbox name.

Microsoft Exchange server: exch.cos.local

Mailbox: Administrator

6. When prompted as to whether you travel with this computer, select **No** and then click **Next**.
7. Click **Finish** and close the remaining dialog boxes.

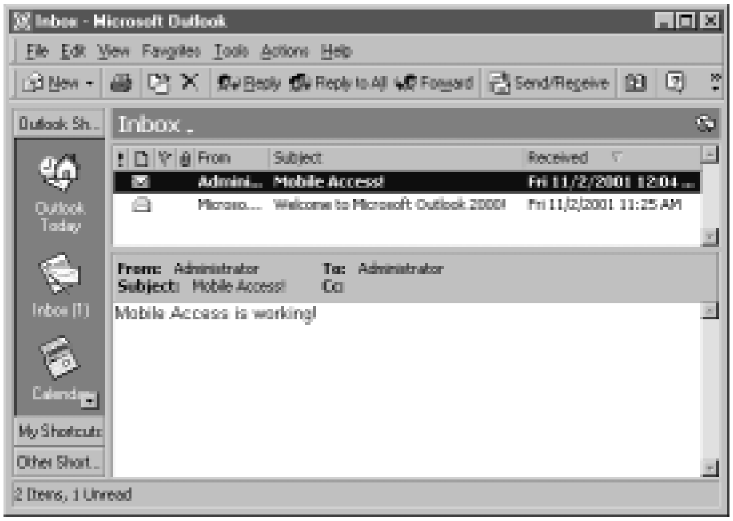
Initializing the Administrator's Mailbox

You are now ready to initialize the administrator's mailbox:

1. Open **Microsoft Outlook** on the Exch machine.

2. If you are prompted for a username and password, use **Administrator** and the administrator's password.
3. Ensure that the mail system is functioning by sending a message to your own account as shown in Figure 7.22.

Figure 7.22 Opening Your Mailbox in Microsoft Outlook Initializes Outlook Mobile Access



From	Subject	Received
Admini...	Mobile Access!	Fri 11/2/2001 12:04 ...
Microso...	Welcome to Microsoft Outlook 2000!	Fri 11/2/2001 11:25 AM

From: Administrator To: Administrator
Subject: Mobile Access! Cc:

Mobile Access is working!

2 Items, 1 Unread

4. Close **Microsoft Outlook**. The mailbox is now initialized for use by Outlook Mobile Access.

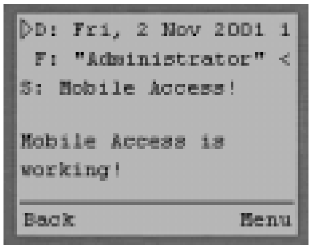
Testing Outlook Mobile Access in a Mobile Phone Simulator

All of the pieces are now in place for mobile access to your mail account. In this lab environment, you can test the mobile phone access by using a software development kit and simulator available from http://developer.openwave.com. For best results, use the 4.1 version of the simulator:

1. Install the mobile phone simulator software on the Exch machine.
2. Run the simulator program, and in the **Go** field, enter **MIS/oma**, where *MIS* is the name of the server that is hosting Mobile Information Server. Do not try to connect to the Exchange server. MIS establishes the link to Exchange Server 2000.

3. When prompted for authentication, enter **Administrator** and the domain administrator's password.
4. Navigate the Outlook Mobile Access pages and verify the messages as shown in Figure 7.23.

Figure 7.23 E-Mail Message on a Mobile Phone Simulator



```
D: Fri, 2 Nov 2001 1
 F: "Administrator" <
S: Mobile Access!

Mobile Access is
working!
```

NOTE

If your Exchange server goes down or is restarted, you may need to restart the Mobile Information Server to re-establish the SMTP connection.

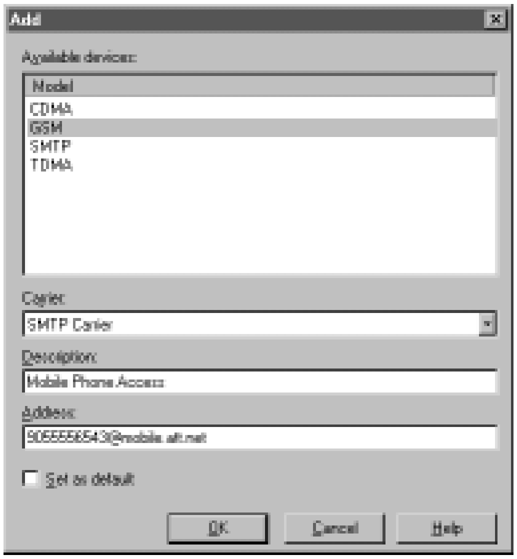
Configuring Users for Outlook Mobile Access

You have now completed the installation and configuration of Mobile Information Server and tested the system as an administrator. Setting up a mailbox for a new user is about the same as configuring a new user in Exchange 2000, except for one additional step:

1. Log on to the Exch machine as a domain administrator.
2. Open **Active Directory Users and Computers**.
3. On the **Users** node, right-click and select **New | User**. Follow the instructions to add a user and mailbox. Throughout this section, the test user is **John Oliver** whose user ID is **olivej**.
4. Open the new user's properties and on the **Member Of** tab, add the user to the **MIS Mobile Users group**.
5. Open the new user's properties page and select the **Wireless Mobility** tab.

6. Check the check box to enable wireless access for this user.
7. Click **Add** and then on the Add page, fill in the details of the device to be used as shown in Figure 7.24.

Figure 7.24 Configuring a Device for Mobile Access by a User



Carrier: SMTP Carrier

Description: Mobile Phone Access

Address: 9055556543@mobile.att.net

☐ Set as default

8. Click **OK** on each dialog box to confirm the creation of the user and mailbox. If prompted, supply (and record) a password for wireless access.
9. From the Outlook client, send the user a test message. Ask the user (John Oliver, in this example) to log in using Outlook to ensure that the mailbox is initialized and receiving mail.
10. As John Oliver, add an **Urgent** folder to the Outlook folders and log out of Outlook. This completes the configuration of a new user.

The main administration chores are now finished. You have installed and configured Microsoft Exchange 2000 as well as Mobile Information Server. Users who have permissions can now use a WAP-enabled device to access their Exchange accounts. In the next section, we look at how users of Outlook Mobile Access can set up their mail and message preferences.

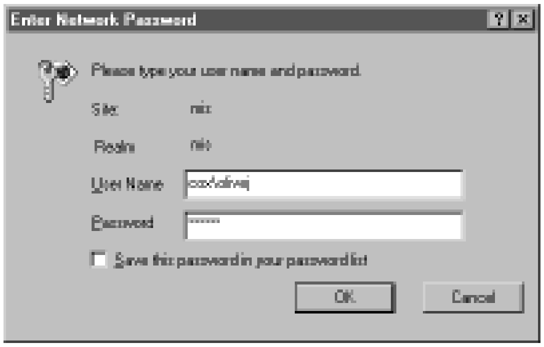
Using Microsoft Mobile Information Server Personalization

Browsing through e-mail with a WAP device such as a mobile phone is not easy. The screen is small and navigation is difficult. Entering alphabetic characters on a keypad that was designed for digits is time-consuming. The slow connection speed adds to the user's burden. For these reasons, users may want to customize their setup to view only certain information via WAP. To allow users to change their settings, Microsoft provides Mobile Information Server personalization pages.

The personalization pages are not available through the WAP device. Users must log in using their Internet Explorer browser. The following instructions show how to use the personalization option:

1. Using Microsoft Internet Explorer, navigate to **http://<*mis_server*>/airweb** where <*mis_server*> is the Mobile Information Server machine. In the examples in this chapter, the URL would be **http://mis/airweb/**.
2. When prompted to log in, type the domain name, a backslash (\), and your Exchange username in the User Name box. Figure 7.25 shows an example of a login.

Figure 7.25 Logging In to the MIS Personalization Web Page



Enter Network Password

Please type your user name and password.

Site: mis

Realm mis

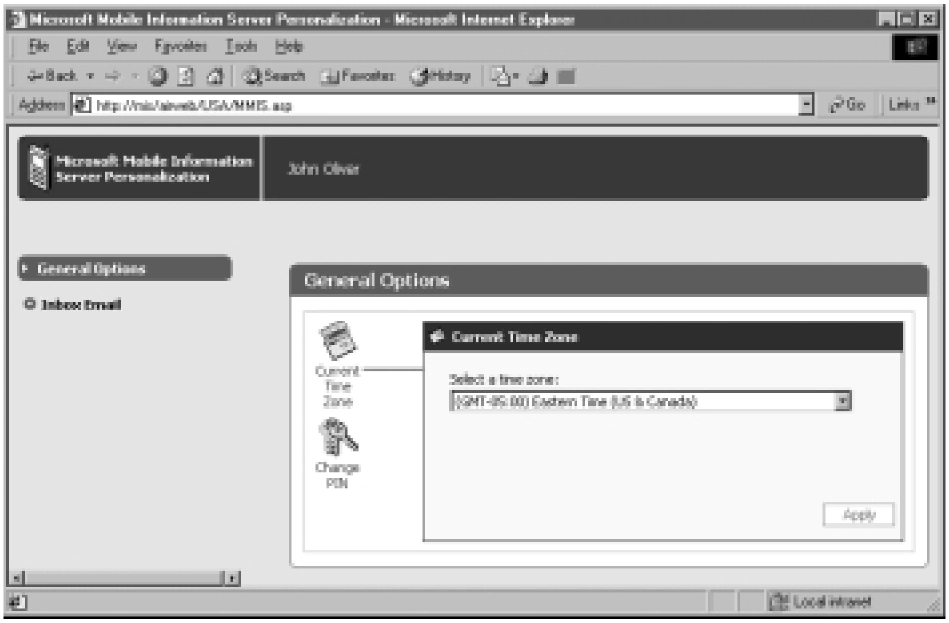
User Name: cox\alfvaj

Password: ******

[ ] Save this password in your password list

3. In the Password box, type your password. Depending on your security settings, you may be prompted for your security credentials a second time for access to the server named Exch. The Personalization page appears, as shown in Figure 7.26.

Figure 7.26 The MIS Personalization Page Showing General Options


Microsoft Mobile Information Server Personalization - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://mis/aiweb/USA/MMIS.asp
Go Links
Microsoft Mobile Information Server Personalization
John Oliver
General Options
Inbox Email
General Options
Current Time Zone
Change PIN
Apply
Local intranet

Current Time Zone
Select a time zone:
(GMT-05:00) Eastern Time (US & Canada)

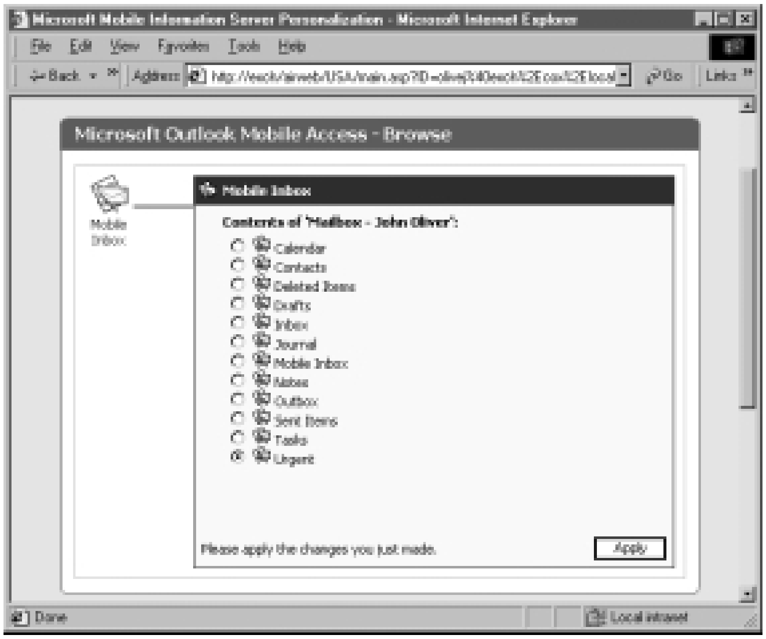
The General Options section lets you select your time zone and change your password. The second category in this example is Inbox E-mail. The actual title of the category depends on the text that the administrator entered when configuring the system.

Changing Your Mailbox Options

You can set one of your Exchange folders as your Mobile Inbox for browsing with Outlook Mobile Access. Let's say that you want direct access to a folder called Urgent. This is where your urgent messages are directed in Outlook. Using the configuration options as shown in Figure 7.27 in the lower part of the screen, you designate the Urgent folder as the Mobile Inbox. The mailbox you choose here will appear as an entry on the main menu of Outlook Mobile Access when you log in from your mobile device.

Using your mobile device, log in to the Outlook Mobile Access site as shown in Figure 7.28. In this case, we logged in using Microsoft Mobile Explorer, another emulator. When you select Mobile Inbox, the Urgent folder appears, as shown in Figure 7.29. You probably want to do a lot more with your messaging, including sending mail directly to your cell phone. We explore that in the next section.

Figure 7.27 Designate a Shortcut to an Exchange Mailbox


Microsoft Mobile Information Server Personalization - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Address Go Links
Microsoft Outlook Mobile Access - Browse
Mobile Inbox
Apply
Done
Local intranet

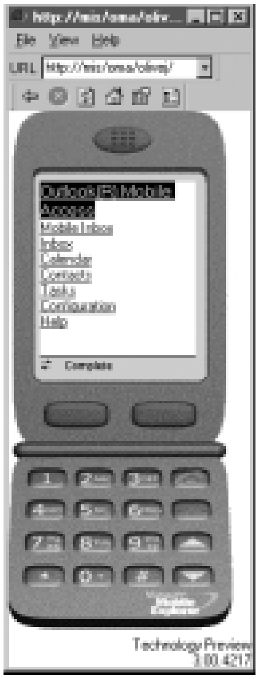
Mobile Inbox

Contents of "Mailbox - John Oliver":

- Calendar
- Contacts
- Deleted Items
- Drafts
- Inbox
- Journal
- Mobile Inbox
- Notes
- Outbox
- Sent Items
- Tasks
- [✓ Urgent]

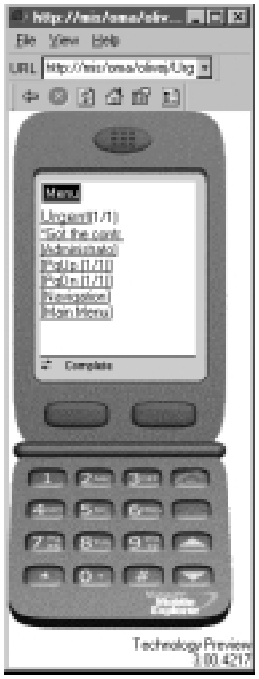
Please apply the changes you just made.

Figure 7.28 The Urgent Folder Is Designated as the Default Folder in Outlook Mobile Access



Outlook(R) Mobile Access
Mobile Inbox
Inbox
Calendar
Contacts
Tasks
Configuration
Help

Figure 7.29 Selecting Mobile Inbox Takes You Directly to the Urgent Folder



NOTE

Only e-mail folders can be set as shortcuts. You cannot set the Calendar, Contacts, Journal, or Tasks as a shortcut.

Configuring Push Notifications

Rather than repeatedly connecting to your inbox to check for messages, you might want to receive alerts on your mobile phone whenever a message arrives that is important to you. In this section, we configure a push notification that matches the following scenario. John Oliver, a mailbox user who has Mobile permissions, wants any messages with the word "Urgent" in them to appear as an e-mail message on his mobile phone. He also wants to keep a copy of the message in his Outlook folders.

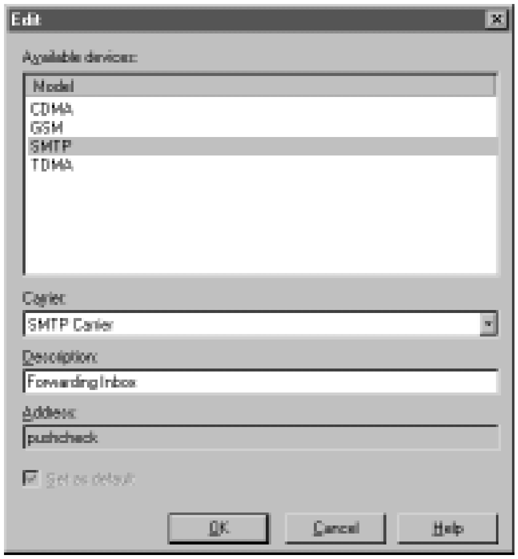
The first task is to add another new mailbox user account in Exchange 2000. This one is the test account that receives the forwarded e-mail message. The goal is to prove that the push is actually forwarding notifications to an e-mail address:

1. On the Exch machine, in Active Directory, add a new usernamed **Push Check**.
2. Make the user's ID **pushcheck**. Exchange automatically adds **@cox.local**.
3. Save your changes to the Push Check user and close the property pages.

The second task is to ensure that John Oliver's configuration is correct in Exchange 2000 and Mobile Information Server:

1. Still on the Exch machine, in Active Directory, open John Oliver's properties.
2. On the **Member Of** tab, make sure that John Oliver is a member of the Mobile Users group.
3. On the **Wireless Mobility** tab, click **Edit** and make sure that pushcheck is listed as the address. Because the e-mail address is local to the domain, don't include the domain name (see Figure 7.30).
4. Save all your changes and exit Active Directory.

Figure 7.30 Add the Address Where the Notification Is to Be Sent


Edit

Available devices:

Model
CDMA
GSM
SMTP
TDMA

Carrier: SMTP Carrier

Description: Forwarding Inbox

Address: pushcheck

☑ Set as default

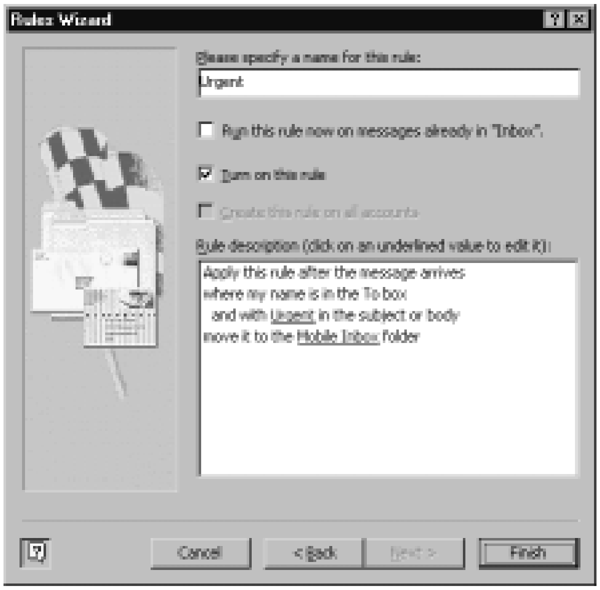
NOTE

You may have a problem changing an existing address. The workaround is to remove the user account from Active Directory (olivej in this case) and re-create it.

Now that you have an address and mailbox account that can "catch" the notification message, you need to configure John Oliver's Outlook client to move all messages containing the word "Urgent" into the Mobile Inbox. The following steps describe how to create a rule in Outlook that functions like the one shown in Figure 7.31:

1. Log in to Microsoft Outlook. In this example we are using the account of **John Oliver**, whose login ID is **olivej**.
2. Select **Tools | Rules Wizard | New**.
3. If you are using Outlook 2002, select the radio button marked **Start from a blank rule**.
4. Select **Check messages when they arrive** and then click **Next**.
5. Check the conditions **where my name is in the To box** and **with specific works in the subject or body**.
6. In the Rule description area, click the link **specific words**.
7. In the **Search Text** dialog box, type **Urgent** as the word to search. Click **Add** and then click **OK**.
8. Back on the Rules Wizard window, click **Next**.
9. Check **move it to the specified folder** and in the lower rule description area, click the **specified** link.
10. In the dialog box, select **Mobile Inbox** and then click **OK**.
11. Click **Next** to skip the exceptions window.
12. Type **Urgent** as the name for the rule and then click **Finish**.
13. Click **OK** to dismiss the Rules Wizard. This completes the creation of the rule.

Figure 7.31 Create a Rule in Outlook to Move Urgent Messages


Rules Wizard

Please specify a name for this rule:

Urgent

☐ Run this rule now on messages already in "Inbox".

☑ Turn on this rule

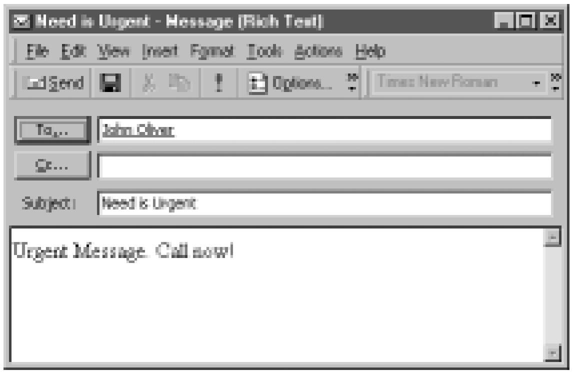
☐ Create this rule on all accounts

Rule description (click on an underlined value to edit it):

Apply this rule after the message arrives
where my name is in the To box
and with Urgent in the subject or body
move it to the Mobile Inbox folder

The last step is to send a message to John Oliver from the administrator's account. Make sure that the message contains the word "Urgent" in the subject line. Figure 7.32 shows a sample message as sent by the Administrator.

Figure 7.32 Send a Message That Contains the Word Urgent


Need is Urgent - Message (Rich Text)

To...: John Oliver

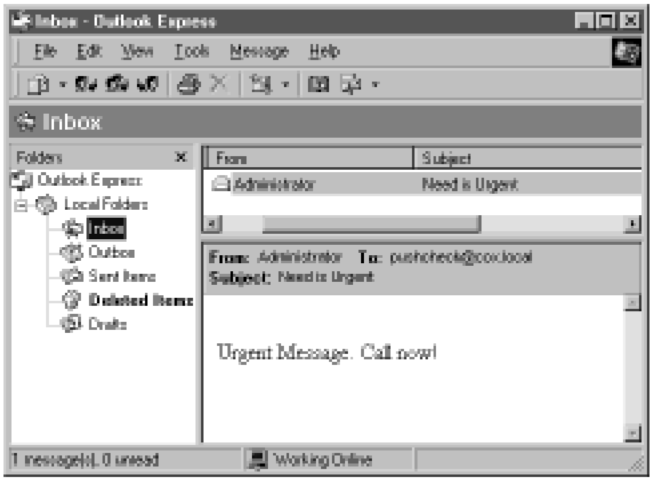
Cc...:

Subject: Need is Urgent

Urgent Message. Call now!

When the message reaches John Oliver's Exchange account, the rule detects the word "Urgent" and moves the message to the Mobile Inbox folder. Exchange and Mobile Information Server pick up whatever arrives in Mobile Inbox, and forward the message to the address listed for the device, which is the user ID pushcheck. If pushcheck were a mobile telephone, the message would appear immediately as an alert. In this case, the notification message appears in the mailbox. Figure 7.33 shows the message in Push Check's Outlook Express client.

Figure 7.33 The Notification Message Appears in the Push Check User's Mailbox



Administrator Need is Urgent

From: Administrator To: pushcheck@cox.local
Subject: Need is Urgent

Urgent Message. Call now!

Although the initial setup of push notification is somewhat intricate, each subsequent addition of an account reuses the same technique with different data.

Browsing the Corporate Intranet with Mobile Information Server

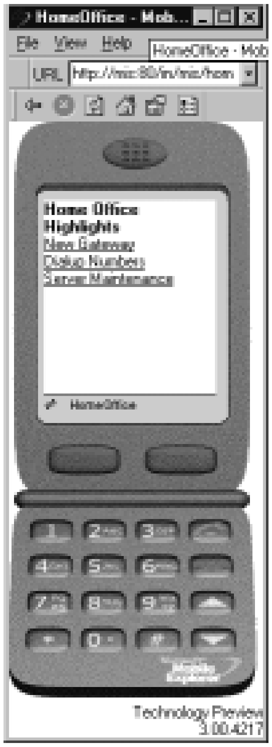
Mobile Information Server allows employees to visit Web sites using their WAP-enabled mobile phone. If the device supports only WML pages, the sites they attempt to visit must support WML. However, some mobile phones may support HTML, which means that employees can visit regular Web pages.

To access an intranet site, users must log in to Mobile Information Server and provide a special URL that includes the server and site that they want to visit. For example, our lab setup includes a virtual directory in Internet Information Server called HomeOffice that includes a Web page. Here is the URL to enter in the device to browse the site:

```
http://mis/in/mis/homeoffice/
```

The URL consists of the MIS server (*mis*) and a special virtual directory name called IN (*in*) followed by the name of the Web server where the content resides (*mis*) and the name of the virtual directory where the page is stored (*home-office*). Figure 7.34 shows how a special intranet site could be set up and viewed by mobile device users.

Figure 7.34 An Intranet Site as Viewed in Microsoft's Mobile Explorer Emulator



Summary of Mobile Information Server Capabilities

You've seen how Mobile Information Server, Outlook Mobile Access and Exchange 2000 combine their functionality to let a user browse his mailbox using a WAP-enabled client. The same software, employing rules added in the Outlook client, allows the user to receive messages as push notifications to the e-mail address of his mobile telephone. In the next section, we look at Microsoft Outlook Mobile Manager (MOMM), another tool that gives users even greater control over their e-mail.

Microsoft Outlook Mobile Manager

Microsoft Outlook Mobile Manager (MOMM) is a free desktop application that gives you advanced control over what Outlook information goes to your text-enabled mobile device such as a cellular phone, pager, or personal digital assistant (PDA). Recognizing the limitations of text messaging on mobile phones, it uses natural language processing to shrink the message content. MOMM provides a convenient, intuitive graphical interface that helps you configure your settings. For example, you can configure MOMM to send reminders from your daily schedule to your mobile device along with your day's tasks.

MOMM is tightly meshed with Mobile Information Server. This section assumes that you have configured MIS as explained in the first part of this chapter and have created and configured the John Oliver (olivej) account as explained in the preceding text. We continue to use a lab environment with one server called Exch that is the domain controller and runs Exchange 2000 and a second server, MIS, that runs Mobile Information Server. Both servers are part of the cox.local laboratory domain. In your environment, the domain might be something like xyzco.com. In addition to the servers, this section uses a client workstation named P733 that runs the Outlook client and Outlook Mobile Manager.

Developing & Deploying…

Choosing between Deployment of Outlook Mobile Access and Outlook Mobile Manager

In a production scenario, supporting users is a major issue and resource drain. Here are some considerations for choosing between Outlook Mobile Access (OMA) and MOMM.

OMA is quite easy to deploy to users because of its simple mobile phone/WAP interface. Users just browse to the Web address on their mobile phone and log in. It doesn't have nearly as many features for notifications, but you can do the configuration over a regular Web browser. You don't have to visit their workstation.

On the other hand, MOMM is a powerful solution that needs to be installed on each user's workstation—a considerable effort in a major

Continued

corporate environment. Because if its richness, "technically challenged" users will likely need some training and support to take advantage of its intricacies.

Also, once people start relying on their remote e-mail messaging, summaries, and notifications, the system's reliability and scalability become important concerns for them—and for you. When people become accustomed to valuable technology, they expect it to work right, all of the time and without excuses.

Whatever way you go, be prepared for an added burden, even if it is just realizing the time it takes to change a user's account in Exchange 2000 server and Active Directory to permit mobile access and initialize the mailbox.

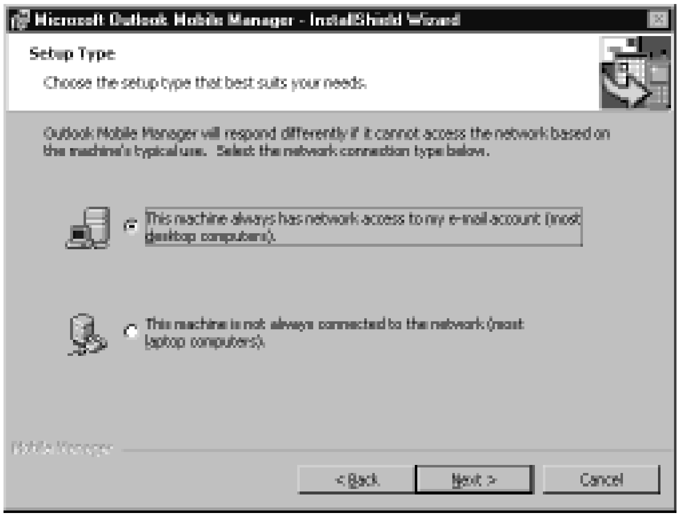
Installing Outlook Mobile Manager

This section shows how to install and configure Outlook Mobile Manager. You can only install Outlook Mobile Manager on a client workstation that has already installed Microsoft Outlook.

In this example, we use the Exchange account of John Oliver, a member of the cox.local domain whose login ID is olivej:

1. Ensure that Microsoft Outlook is installed but *not* running on the client machine and then run the Outlook Mobile Manager installation executable.
2. Step through the Welcome, License Agreement, and Customer Information by filling in the requested information and clicking **Next**. For best results, choose the option to install for all users of the computer.
3. As shown in Figure 7.35, on the **Setup Type** screen, choose the option for desktop computers that always have network access and then click **Next**.
4. Accept the default installation folder and click **Next.** Click **Install**.
5. Check the option to start Outlook Mobile Manager now and click **Finish**. This completes the installation of the software. The next section shows you how to configure the settings.

Figure 7.35 Select the Option for a Desktop Computer with Constant Network Access


Microsoft Outlook Mobile Manager - InstallShield Wizard

Setup Type

Choose the setup type that best suits your needs.

Outlook Mobile Manager will respond differently if it cannot access the network based on the machine's typical use. Select the network connection type below.

(●) This machine always has network access to my e-mail account (most desktop computers).

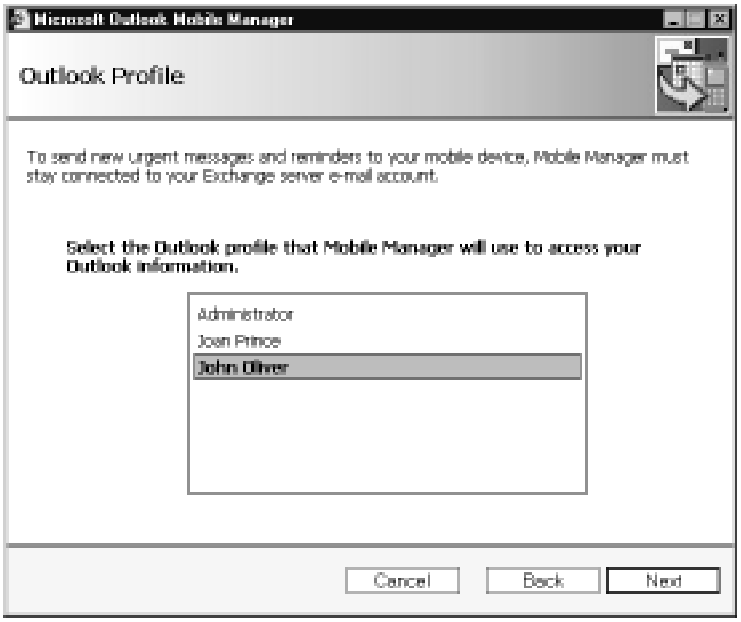
( ) This machine is not always connected to the network (most laptop computers).

Configuring Outlook Mobile Manager

A connection wizard walks you through the configuration of Outlook Mobile Manager. Because this is a lab environment, we configure the system to forward messages to a local e-mail account. However, in a production application, messages go to the e-mail address of the mobile device:

1. Run **Outlook Mobile Manager** and on the Welcome screen click **Next**.
2. Select the Outlook Profile that Mobile Manager will use as its source of information. In this case, we are configuring for user John Oliver, as shown in Figure 7.36. Click **Next**.
3. When prompted for the type of connection, select a company-based Mobile Information Server, as shown in Figure 7.37, and then click **Next**.
4. When prompted for the username, type the Windows networking or Exchange e-mail ID. In this lab scenario for user John Oliver, type **olivej** and then click **Next**.

Figure 7.36 Select a Profile that Outlook Mobile Manager Will Use


Microsoft Outlook Mobile Manager

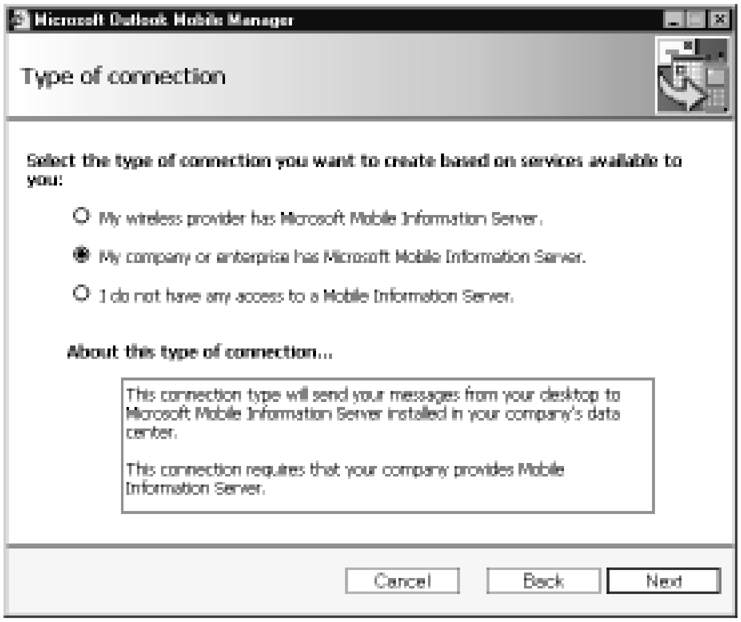
Outlook Profile

To send new urgent messages and reminders to your mobile device, Mobile Manager must stay connected to your Exchange server e-mail account.

Select the Outlook profile that Mobile Manager will use to access your Outlook information.

Administrator
Joan Prince
John Oliver

Figure 7.37 In a Corporate Environment or Test Lab, Choose the Company's Mobile Information Server


Microsoft Outlook Mobile Manager

Type of connection

Select the type of connection you want to create based on services available to you:

- ○ My wireless provider has Microsoft Mobile Information Server.
- ● My company or enterprise has Microsoft Mobile Information Server.
- ○ I do not have any access to a Mobile Information Server.

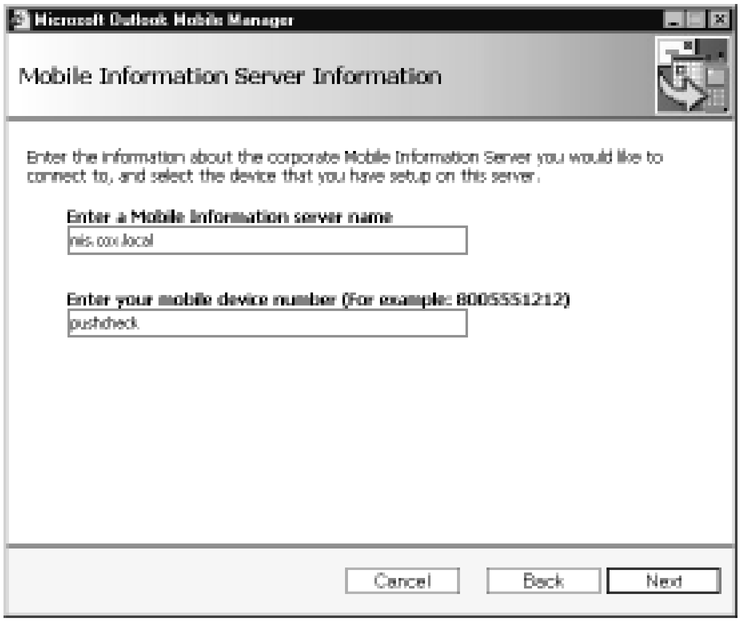
About this type of connection...

This connection type will send your messages from your desktop to Microsoft Mobile Information Server installed in your company's data center.

This connection requires that your company provides Mobile Information Server.

5. In the **Mobile Information Server** name box, type the name of the server including its domain. In a production scenario, you would enter the number of the mobile device (Figure 7.38). In the lab scenario, enter an Exchange mailbox ID.

Figure 7.38 In the Lab Environment, the Mobile Device Number Is a Local E-Mail Account


Microsoft Outlook Mobile Manager
Mobile Information Server Information

Enter the information about the corporate Mobile Information Server you would like to connect to, and select the device that you have setup on this server.

Enter a Mobile Information server name
mis.cxx.local

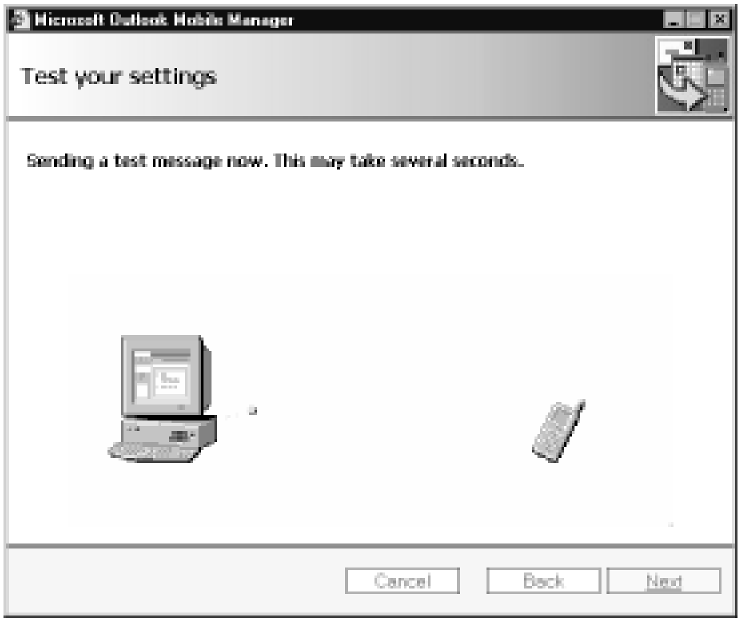
Enter your mobile device number (For example: 8005551212)
pushcheck

6. On the Test screen, click **Next**. The configuration tool sends a test message to the target device, as shown in Figure 7.39.
7. If the test is successful, the configuration routine displays the results, as shown in Figure 7.40. Click **Finish**.

Understanding Outlook Mobile Manager

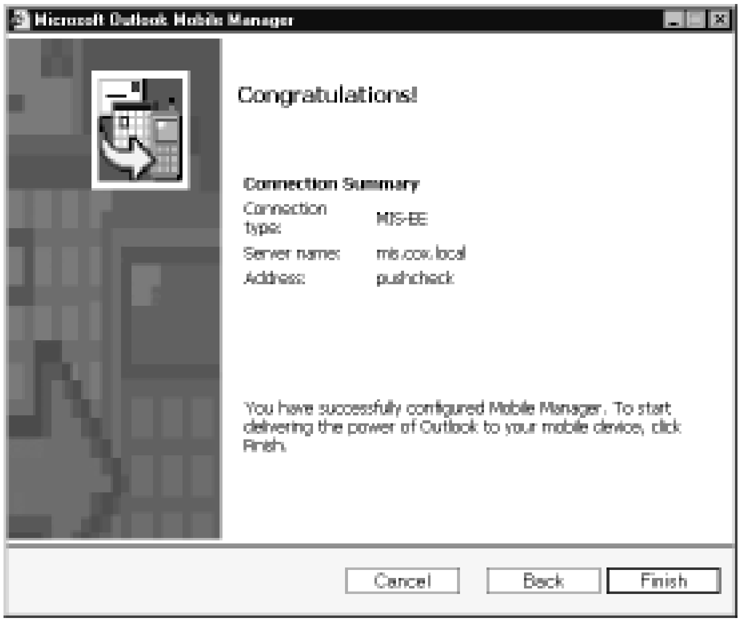
After installing Outlook Mobile Manager successfully, you need to log into the same account that you chose (olivej) during the setup phase. Notice that a new mobile device icon appears in your desktop computer's system tray. The user interface for MOMM, as shown in Figure 7.41, is quite intuitive. We cover the main features and tasks.

Figure 7.39 The Configuration Tool Sends a Test Message via Mobile Information Server



Sending a test message now. This may take several seconds.

Figure 7.40 A Successful Test Shows the Connection and Destination Information

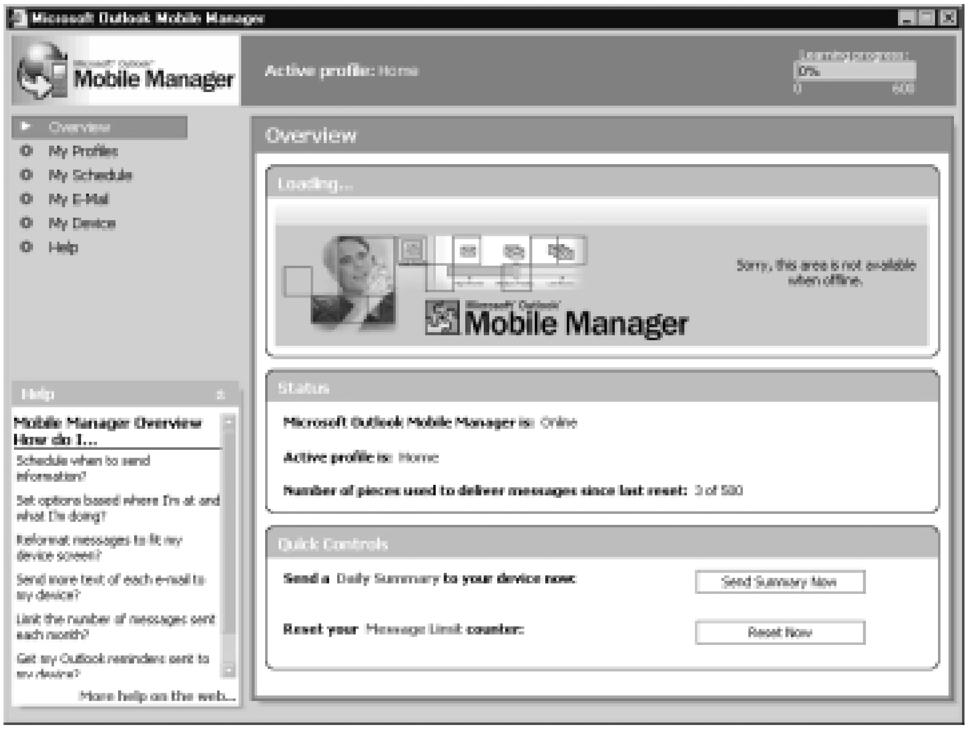


Connection Summary

Connection type:	MIS-BE
Server name:	mis.cox.local
Address:	pushcheck

You have successfully configured Mobile Manager. To start delivering the power of Outlook to your mobile device, click Finish.

Figure 7.41 The Startup User Interface for Outlook Mobile Manager



Active profile: Home

Sorry, this area is not available when offline.

Mobile Manager Overview
How do I...
Schedule when to send information?
Set options based where I'm at and what I'm doing?
Reformat messages to fit my device screen?
Send more text of each e-mail to my device?
Limit the number of messages sent each month?
Get my Outlook reminders sent to my device?

Microsoft Outlook Mobile Manager is: Online

Active profile is: Home

Number of pieces used to deliver messages since last reset: 3 of 500

Send a Daily Summary **to your device now:**

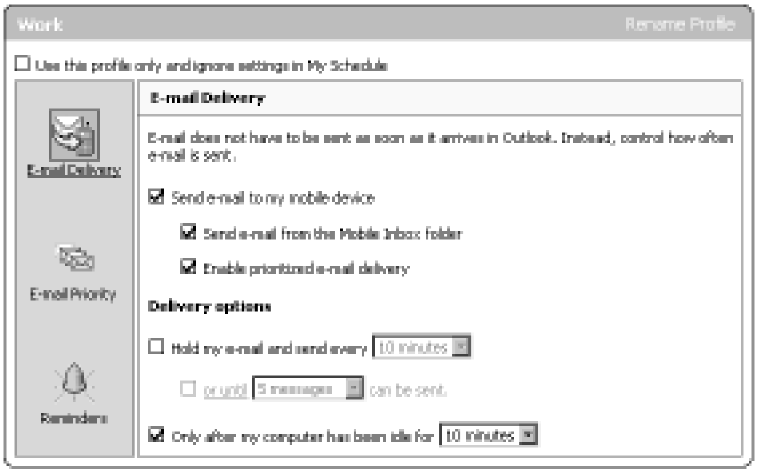
Reset your Message Limit **counter:**

Configuring a Profile

Profiles let you configure levels of message forwarding activity depending on your situation. For example, when at work, you might want most e-mail messages and all reminders to be forwarded to your mobile device. On the weekend when you are at home, you would want only urgent messages. While on vacation, you could choose your Do Not Disturb profile so that no messages or alerts would be sent to your cell phone. You can customize each option. The choices include the type of messages sent, the priority, and the number of messages sent in a given period.

Figure 7.42 shows the default settings for John Oliver's Work profile. Rather than forwarding every message to the cell phone, Outlook Mobile Manager sends only messages that arrive in the Mobile Inbox folder. In the earlier section called "Configuring Push Notifications," you saw how to create an Outlook rule that forwards only certain messages to the Mobile Inbox. Messages are also forwarded based on a priority that you'll see in a moment. Also, messages may not be sent to the mobile device while the desktop computer is occupied with other tasks.

Figure 7.42 E-Mail Delivery Options in the Work Profile



☐ Use this profile only and ignore settings in My Schedule

E-mail Delivery

E-mail does not have to be sent as soon as it arrives in Outlook. Instead, control how often e-mail is sent.

☑ Send e-mail to my mobile device

☑ Send e-mail from the Mobile Inbox folder

☑ Enable prioritized e-mail delivery

Delivery options

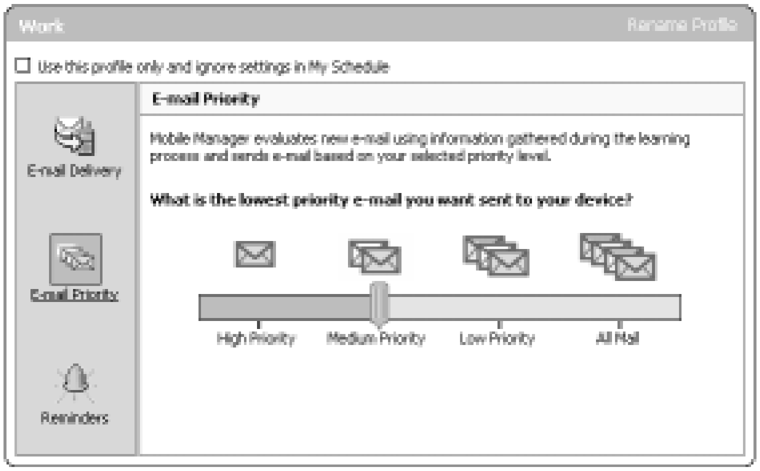
☐ Hold my e-mail and send every 10 minutes

☐ or until 5 messages can be sent.

☑ Only after my computer has been idle for 10 minutes

You probably don't want all messages sent to your mobile phone, even in your Work profile. You can set an option to allow only high priority messages as determined by the priority flag on the message and other characteristics. Figure 7.43 shows the e-mail priority at a medium setting.

Figure 7.43 Configuring the Message Delivery Priority



☐ Use this profile only and ignore settings in My Schedule

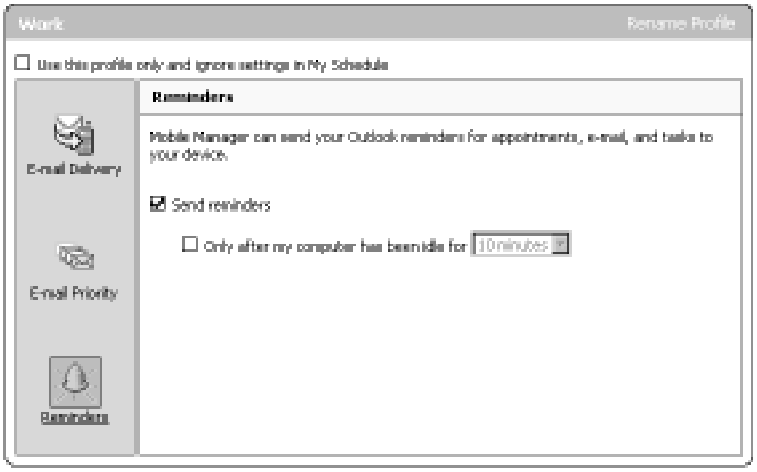
E-mail Priority

Mobile Manager evaluates new e-mail using information gathered during the learning process and sends e-mail based on your selected priority level.

What is the lowest priority e-mail you want sent to your device?

The Reminders option sends e-mail alerts whenever Outlook would send an appointment or task alert reminder to the desktop. In Figure 7.44, reminders are sent to the mobile phone immediately, whether or not the computer is idle.

Figure 7.44 Setting Outlook Mobile Manager to Send Reminders of Appointments and Tasks



Work Rename Profile

☐ Use this profile only and ignore settings in My Schedule

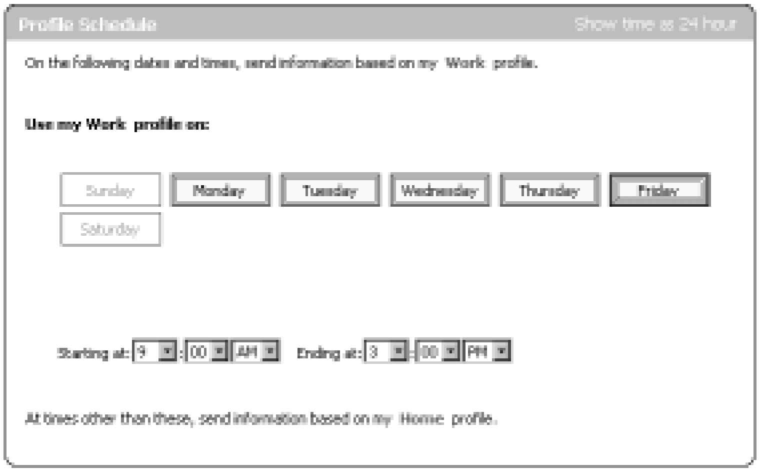
Reminders

Mobile Manager can send your Outlook reminders for appointments, e-mail, and tasks to your device.

☑ Send reminders

☐ Only after my computer has been idle for 10 minutes

Figure 7.45 Scheduling to Change from the Work Profile to Home Profile at 3 P.M. on Fridays



Profile Schedule Show time as 24 hour

On the following dates and times, send information based on my Work profile.

Use my Work profile on:

Sunday | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday

Starting at: 9 : 00 AM Ending at: 3 : 00 PM

At times other than these, send information based on my Home profile.

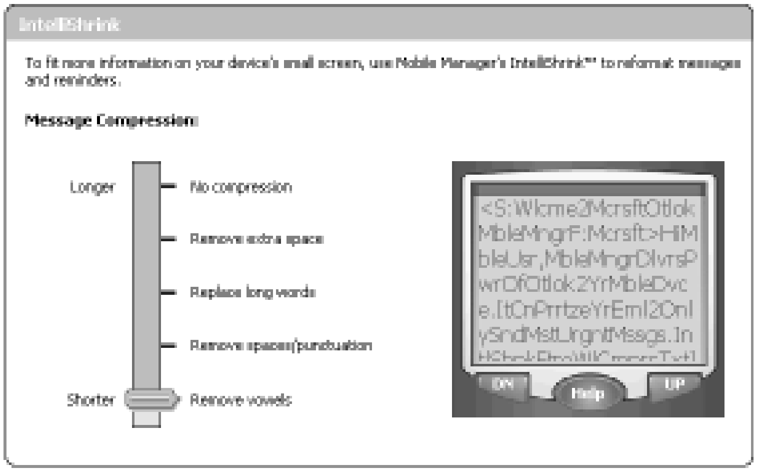
Manually changing from the Work profile to the Home profile each day at quitting time and before the weekend would be a nuisance. Therefore, MOMM lets you create a schedule for automatically shifting from one to profile to the other. Figure 7.45 shows a schedule in which John Oliver's profile is Work from 9 to 3 Monday through Friday (he goes home early!). After that the profile automatically shifts to Home. The Home profile remains in effect throughout the weekend.

Compressing the Size of Text Messages

Typically, mobile devices have limited bandwidth and very little room on the display for reading text. Not only do you want to reduce the number of messages to the essential, you also want to make the messages as brief as possible. Outlook Mobile Manager allows you to set a compression level for messages. At one extreme, nothing is removed from the messages. At the other, all vowels, spaces and formatting characters are removed. To test the IntelliShrink feature, follow these steps:

1. In the upper-left navigation area of Outlook Mobile Manager, click **My Device** and then click **IntelliShrink**.
2. In the **My Device** box, move the slider to the bottom of its range to **Shorter** as shown in Figure 7.46. This is the extreme case, which will remove extra spaces, long words, and vowels.

Figure 7.46 IntelliShrink Removes Extraneous Text and Shortens Words



IntelliShrink

To fit more information on your device's small screen, use Mobile Manager's IntelliShrink™ to reformat messages and reminders.

Message Compression:

Longer — No compression
Remove extra space
Replace long words
Remove spaces/punctuation
Shorter — Remove vowels

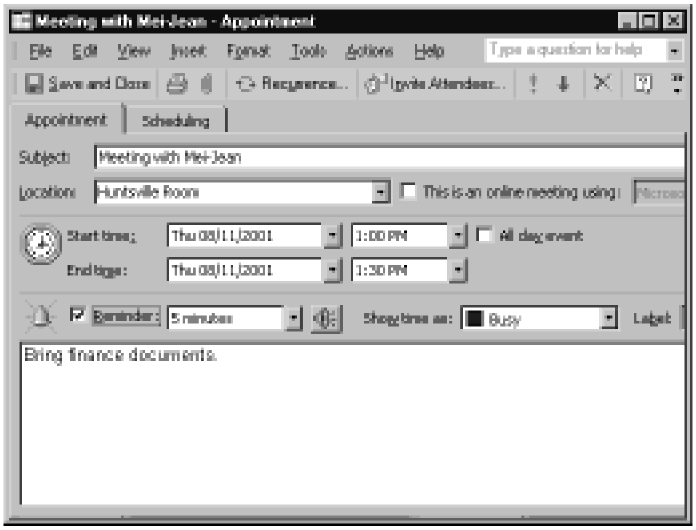
<S:Wlcme2McrsftOtlok
MbleMngrF:Mcrsft>HiM
bleUsr,MbleMngrDlvrsP
wrOfOtlok2YrMbleDvc
e.ItCnPrrtzeYrEml2Onl
ySndMstUrgntMssgs.In

Sending a Reminder to Your Mobile Device

You can configure Outlook Mobile Manager to automatically send a reminder for appointments to your mobile device. Follow these steps to configure a reminder:

1. In Outlook, add an appointment to your calendar.
2. Set the reminder, as shown in Figure 7.47, so that it will alert you within a few minutes.

Figure 7.47 Adding an Outlook Appointment with a Reminder



Subject: Meeting with Mei-Jean

Location: Huntsville Room ☐ This is an online meeting using:

Start time: Thu 08/11/2001 1:00 PM ☐ All day event

End time: Thu 08/11/2001 1:30 PM

☑ Reminder: 5 minutes Show time as: Busy Label:

Bring finance documents.

3. Save and close the appointment.
4. In **Outlook Mobile Manager**, make sure that **Send reminders** is checked as shown in Figure 7.44.
5. After the reminder period has passed, check the device for the notification.

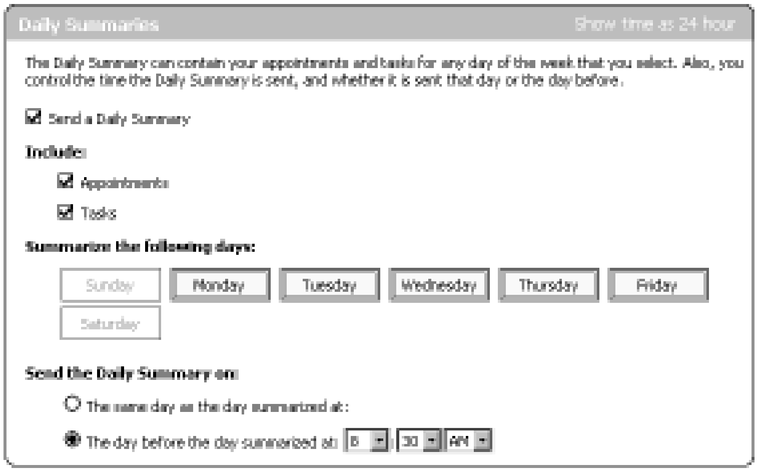
In the lab, we used a local e-mail account rather than a cell phone to receive and view the reminder. The following text shows the message that was received via Mobile Information Server:

```
<RMD:Meeting with Mei-Jean
1:00P-1:30P:Huntsville Room
F:<olivej@cox.local>>
Bring finance documents.
```

You can also configure Outlook Mobile Manager to send you a summary of the day's events as found in your Outlook Schedule and Tasks. Figure 7.48 shows the configuration of the Daily Summary page to send a preview of the day's appointments and tasks at 8:30 in the morning.

As you've seen, Outlook Mobile Manager is a comprehensive tool for managing and scheduling messages to your mobile device. Although it offers a rich array of configuration choices, Outlook Mobile Manager has the disadvantage of being a desktop application. You must be at—or connected to—your workstation to make changes to your configuration.

Figure 7.48 Configure Outlook Mobile Manager to Send a Daily Summary of Events



Daily Summaries   Show time as 24 hour

The Daily Summary can contain your appointments and tasks for any day of the week that you select. Also, you control the time the Daily Summary is sent, and whether it is sent that day or the day before.

[✓] Send a Daily Summary

Include:

[✓] Appointments

[✓] Tasks

Summarize the following days:

Sunday Monday Tuesday Wednesday Thursday Friday Saturday

Send the Daily Summary on:

○ The same day as the day summarized at:

● The day before the day summarized at: 8 : 30 AM

Clearly, Microsoft is preparing for a generation of rich mobile devices that go beyond the limited capabilities of many current WAP-enabled cellular telephones. The new devices, based on the Pocket PC and Windows CE technologies, will bring wireless mobile devices closer to the ease of use that you expect and enjoy on the desktop. In that context, the next section shifts to a mobile solution using the Pocket PC and Microsoft SQL Server 2000 for Windows CE.

Microsoft SQL Server 2000 CE Edition

One way or another, most computer applications end up consuming, storing, and rendering data. Computers can store data in many formats, but the relational database system is the most powerful and efficient for complex data queries. To provide small devices with a robust database system, Microsoft has introduced SQL Server 2000 CE Edition. Even with its small footprint, SQL Server 2000 CE Edition supports the fundamental SQL Server programming features. It provides advanced capabilities including remote data access, merge replication, and support for transactions. In this section, we look at installing and using some basic features of SQL Server 2000 CE on a Pocket PC, specifically the Compaq iPAQ. Not every mobile application needs to be in constant communication with a home base as long as the required data is portable and secure. We look at how you can "pull" data from a SQL Server 2000 database to your Pocket PC and view it instantly offline.

Installing Microsoft SQL Server 2000 Windows CE Edition

Although the installation of SQL Server CE is relatively easy, configuring it requires knowledge not only of its big brother, SQL Server 2000, but also of Internet Information Server. That's because the CE version doesn't deal directly with the master database but communicates via the Web server. We start with the installation and configuration of the server, then do the installation on the development workstation and finally move on to the Web server, which can run on the same server or a separate machine.

To develop applications for SQL Server 2000 CE Edition, you need a fair amount of software. Table 7.1 shows the machines involved and the software that each requires.

Table 7.1 Software Requirements to Develop SQL Server CE Applications

Machine	Software Required
Server	SQL Server 2000 SQL Server 2000 Windows CE Edition Server Tools Internet Information Server (may be on a separate machine)
Workstation	SQL Server 2000 Developer Edition SQL Server 2000 Windows CD Edition Development Tools Microsoft ActiveSync Microsoft eMbedded Visual Tools 3.0 Windows CE SDK
Pocket PC	SQL Server CE component (automatically installed by software that uses it)

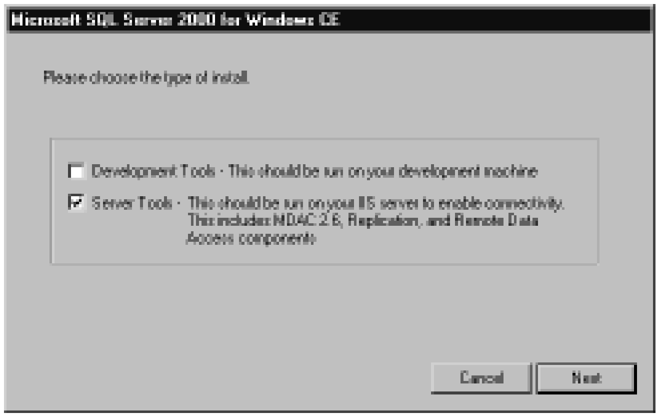
In this scenario, we run the developer edition of SQL Server 2000 as well as IIS on the same computer. The development environment runs on a separate workstation. The Pocket PC is connected to the workstation via a USB cable, and that connection is managed by Microsoft's ActiveSync software.

Installing SQL Server CE on the SQL/IIS Server

This section assumes that you have already installed SQL Server 2000 and IIS on the server and SQL Server 2000 Developer Edition on the development workstation. Follow these steps to install SQL Server 2000 CE Edition on the server:

1. On the IIS/SQL Server machine, run the setup program for SQL Server 2000 Windows CE Edition.
2. As shown in Figure 7.49, check the **Server Tools** option and then click **Next**.

Figure 7.49 Select the Server Tools Option When Installing SQL Server 2000 for Windows CE



Microsoft SQL Server 2000 for Windows CE

Please choose the type of install.

☐ Development Tools - This should be run on your development machine

☑ Server Tools - This should be run on your IIS server to enable connectivity. This includes MDAC 2.6, Replication, and Remote Data Access components

3. On the remaining screens (**Welcome**, **Installation** folder, and **Confirmation**), click **Next** to begin the installation.
4. On the **Installation Complete** screen, click **Close**. This completes the installation of the SQL Server CE components on the server.

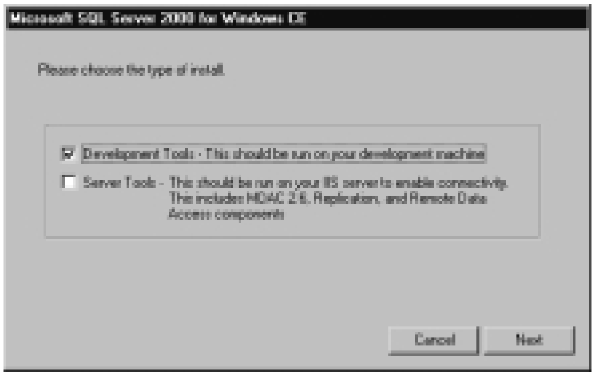
Installing SQL Server CE on the Development Workstation

Before you can do development work with SQL Server 2000 Windows CE Edition on your workstation, you need to install several other pieces of software. If you haven't already done so, check the requirements in Table 7.1. This installation assumes that the other parts are installed. Your Pocket PC or other Windows CE device should be connected to your development workstation. However, you can use the Pocket PC emulator that is included in the Visual Tools package:

1. On your Windows 2000 development workstation, run the setup program for SQL Server 2000 Windows CE Edition. This installation includes some remote data access components.

2. When prompted for the installation type, select **Development Tools**, as shown in Figure 7.50.

Figure 7.50 On the Workstation, Install the Development Tools Option of SQL Server 2000 CE



Microsoft SQL Server 2000 for Windows CE

Please choose the type of install.

[✓] Development Tools - This should be run on your development machine

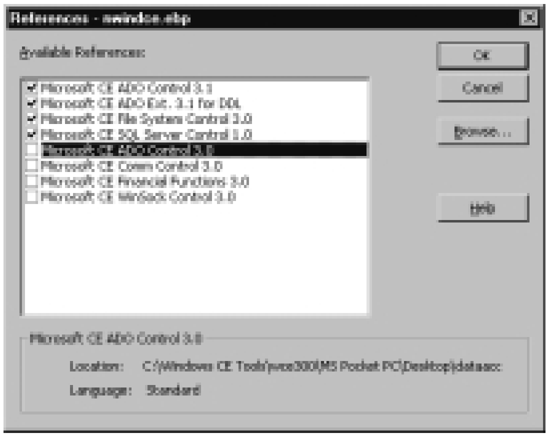
[ ] Server Tools - This should be run on your IIS server to enable connectivity. This includes MDAC 2.6, Replication, and Remote Data Access components

3. Complete the installation steps as you did on the server.

Installing SQL Server CE on the Pocket PC

The Pocket PC installation is the easiest of all and is handled automatically by software that requires it—provided that the application developer has included the correct settings. When creating a program for SQL CE, set the project properties to update project components. Also include a reference to the SQL Server CE component as shown in Figure 7.51.

Figure 7.51 The SQL CE Server Control is Installed with Software That Requires It



References - nwindce.ebp

Available References:

[✓] Microsoft CE ADO Control 3.1
[✓] Microsoft CE ADO Ext. 3.1 for DDL
[✓] Microsoft CE File System Control 3.0
[✓] Microsoft CE SQL Server Control 1.0
[ ] Microsoft CE ADO Control 3.0
[ ] Microsoft CE Comm Control 3.0
[ ] Microsoft CE Financial Functions 3.0
[ ] Microsoft CE WinSock Control 3.0

Microsoft CE ADO Control 3.0

Location: C:\Windows CE Tools\wce300\MS Pocket PC\Desktop\dataacc

Language: Standard

The installation tool (**Tools | Remote Tools | Application Install Wizard**) will then package up the required executables into a CAB file. You just need to download or copy the CAB file to your Pocket PC and click on the CAB file. The Windows CE utilities take care of installing and registering the necessary parts, including the vital SQL Server CE component.

Configuring Internet Information Server for SQL Server CE Use

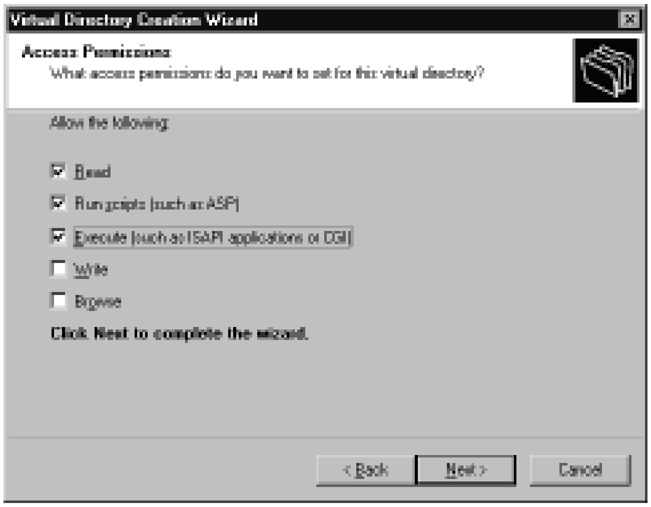
As mentioned previously, SQL Server CE on the Pocket PC talks to the master SQL Server database indirectly through the HTTP protocol on Internet Information Server. The executable program that handles this communication is an IIS plug-in called the SQL Server CE Server Agent ISAPI DLL. The following steps show how to configure the Web server:

1. On the IIS/SQL Server machine, using Windows Explorer, create a new directory called **Northwind** in the Web server's root directory. In the default scenario, this directory will be **c:\inetpub\wwwroot\northwind**.
2. Using Windows Explorer, navigate to the directory **C:\Program Files\Microsoft SQL Server CE\Server** and locate the file **sscesa10.dll**.
3. Copy **sscesa10.dll** into the directory that you created in Step 1: c:\inetpub\wwwroot\northwind.
4. From a command prompt, register sscesa10.dll using the following command:

   ```
   regsvr32 c:\inetpub\wwwroot\northwind\sscesa10.dll
   ```

5. Open Internet Services Manager and expand the nodes to display the Default Web Site node.
6. Right-click on **Default Web Site** and from the context menu, click **New | Virtual Directory**.
7. Follow the steps in the Virtual Directory Wizard. Type **northwind** as the alias, browse to the folder that you previously created (c:\inetpub\wwwroot\northwind), and on the Access Permissions window, make sure that **Execute** is checked, as shown in Figure 7.52.

Figure 7.52 Ensure That Execute Permissions Are Allowed on the Virtual Directory


Virtual Directory Creation Wizard

Access Permissions
What access permissions do you want to set for this virtual directory?

Allow the following:

- [x] Read
- [x] Run scripts (such as ASP)
- [x] Execute (such as ISAPI applications or CGI)
- [ ] Write
- [ ] Browse

Click Next to complete the wizard.

8. Finish the Virtual Directory Creation Wizard's steps and exit Internet Information Services Manager. This completes the virtual directory configuration.

Debugging…

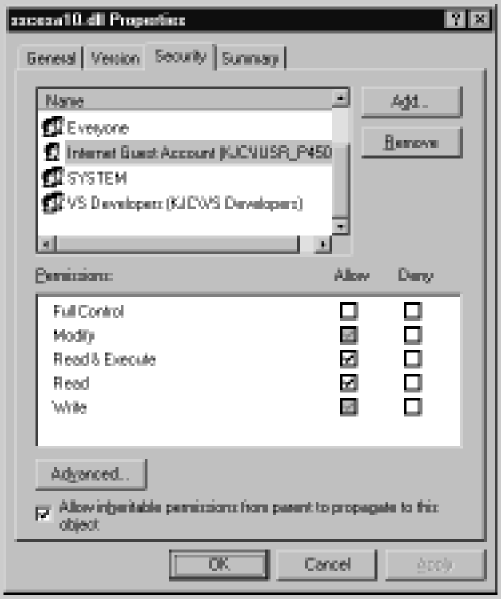
Verify the Web Connection to SQL Server

Connectivity is the biggest issue with SQL CE, and permissions are crucial to the success of this installation. If your connections aren't working, try the following steps to analyze and debug the problem:

1. Using Windows Explorer, navigate to the folder where you installed sscesa10.dll: (c:\inetpub\wwwroot\northwind).
2. Right-click on **sscesa10.dll** and from the context menu, click **Properties**.
3. On the Security tab, make sure that **Read** and **Execute** are checked for Internet Guest Account as shown in Figure 7.53. If the Internet Guess Account does not appear, add it. Close the property page when you are done.

Continued

Figure 7.53 The Internet Guest Account Requires Execute Rights in the File System



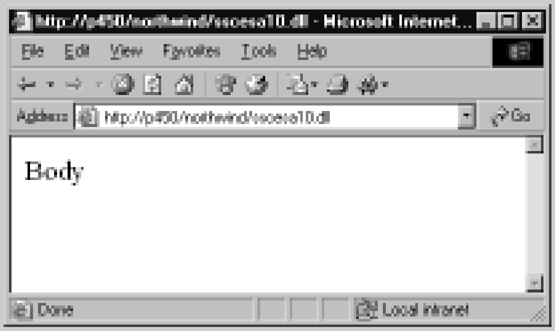
Name
Everyone
Internet Guest Account (KJC\IUSR_P450
SYSTEM
VS Developers (KJC\VS Developers)

Permissions:	Allow	Deny
Full Control	☐	☐
Modify	▨	☐
Read & Execute	✓	☐
Read	✓	☐
Write	▨	☐

✓ Allow inheritable permissions from parent to propagate to this object

4. Test your IIS setup by using Internet Explorer to browse to the Web location of sscesa10.dll as shown in Figure 7.54. You should not get an error message when you open the ISAPI DLL in your browser. Check that sscesa10.dll is registered.

Figure 7.54 Browsing to Verify the Installation of the ISAPI DLL



http://p450/northwind/sscesa10.dll

Body

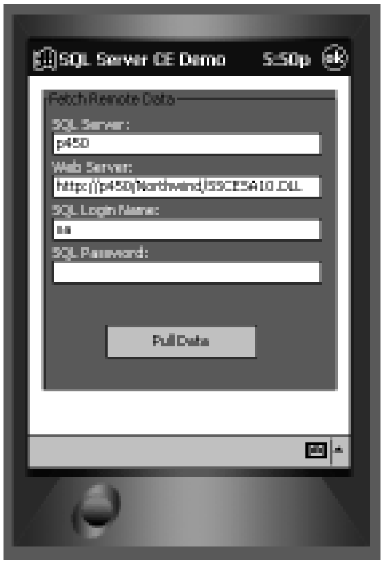
If the system stops working at a later date, check whether someone has run a security tool or other wizard that has clamped down permissions so tightly that data can no longer be passed properly.

Pulling Data from SQL Server to SQL Server CE on the Pocket PC

In this section, we create a sample application that pulls data from a remote SQL Server and stores it on a Pocket PC running SQL Server CE. This is just a proof-of-concept application. It does not do additions or updates to the data. Replication—because it requires advanced SQL programming, more configuration steps, and specific security settings—is beyond the scope of this chapter You can find the source code for the demonstration program at **www.syngress.com/solutions**. Look for the project file called nwindce.ebp and the form code file called form1.ebf.

You can imagine a scenario where you want to take corporate data on the road—but only selected data that pertains to your clients. Using remote data access, you can retrieve the relevant records from the master database, disconnect, and then browse the data offline. Figure 7.55 shows the startup page of the SQL Server CE Demo application built in Microsoft eMbedded Visual Basic 3.0. While connected to the workstation using a USB cable and ActiveSync, you fetch remote data from the SQL Server (p450) machine. Notice that the connection requires the URL of the Web server because the Web server communicates with SQL Server on your behalf. It also requires a user ID and password for SQL Server.

Figure 7.55 At the Login Screen, the Pocket PC Pulls Data from the SQL Server Using IIS



SQL Server: p450
Web Server: http://p450/Northwind/SSCESA10.DLL
SQL Login Name: sa
SQL Password:

NOTE

The examples in this section—and in the chapter as a whole—are intended for a lab environment. To keep the deployment as simple as possible, we left permissions wide-open. In a production environment, you would gradually tighten security as you test your application.

When you click **Pull Data**, the program creates a connection to the remote database using the *SSCE.RemoteDataAccess.1.0* (RDA) object. Using properties of this object, specify the connection string for the data provider name and the Internet URL for the server agent (IIS).

This project uses the RDA object's *Pull* method to fetch the remote data. The sample call, as found in Form1.ebp at **www.syngress.com/solutions**, would look like the following:

SYNGRESS syngress.com

```
On Error Resume Next
Set rda = CreateObject("SSCE.RemoteDataAccess.1.0")
rda.LocalConnectionString = DataProvider & PathandFile
rda.InternetURL = URL
rda.Pull CETable, SQLString, SQLServerConn, TRACKINGON, CEErrorTable
If rda.ErrorRecords.Count > 0 Then
  Call ShowErrors(rda.ErrorRecords)
End If
Set rda = Nothing
```

Table 7.2 shows the parameters that the *Pull* method takes and describes the purpose of the data.

Table 7.2 Parameters Used by the RDA Object's *Pull* Method

Parameter	Description
CETable	This is the name of the SQL Server CE table on the Pocket PC. This table holds the records that are pulled from the remote SQL Server. In the current project, the table name is *locCustomers*.
SQLString	The string that specifies which rows and columns to pull from the remote database. For example, in the current project, this is the following query:

Continued

Table 7.2 Continued

Parameter	Description
	SELECT CustomerID, CompanyName, Address, City, Phone FROM Customers.
SQLServerConn	The OLE DB connection string used to connect to the SQL Server database. The string in this project is: *Provider=sqloledb;Initial Catalog=Northwind; "Data Source=p450;UID=sa;password=".* The preceding string indicates that the connection should be made to the Northwind database on the server called p450 using *sa* as the user ID and a blank password.
TRACKINGOn	The tracking option determines whether the SQL Server CE database on the Pocket PC should track changes made to the data. A value of 1 turns on tracking; a value of 0 means no tracking.
CEErrorTable	This parameter identifies the local table for tracking errors.

After executing a *Pull*, the Pocket PC has its own copy of the data and disconnects from the remote database. At this point, the user can browse the local version as shown in Figures 7.56 and 7.57.

Figure 7.56 The User Can Select Data from the Disconnected Records

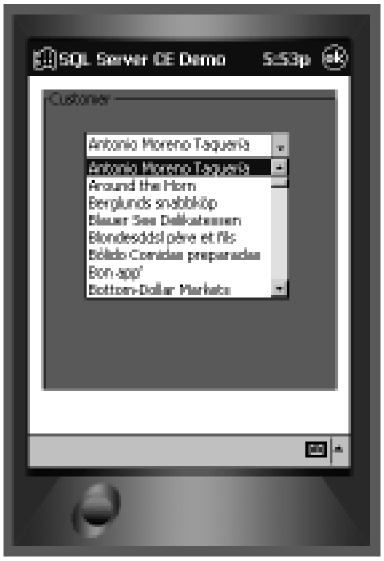
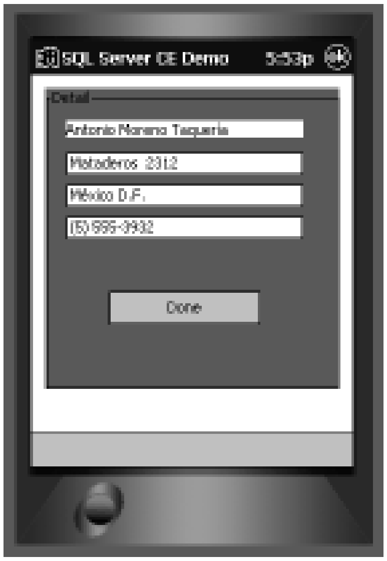


Figure 7.57 SQL Server CE Allows Filtering and Browsing of Data



Antonio Moreno Taquería
Mataderos 2312
México D.F.
(5) 555-3932

Using Replication and Merging for Advanced Functionality

With replication and merging, you can imagine many practical and powerful applications for this database technology. Sales agents, meter readers, or doctors can download customer and patient records in the morning and then review, collect, and update data on their portable devices throughout the day. Back at the workstation, the person returns the Pocket PC to its cradle where ActiveSync automatically creates a connection. One click of an Upload button sends the day's data to the master database for processing and storage. There's no additional effort or retyping that can introduce errors.

Summary

This chapter presented leading-edge technologies for fetching, viewing, and interacting with messages and data from remote servers by means of wireless and remote devices. You saw how to install Mobile Information Server and configure it to access to an Exchange Inbox on mobile telephones. You saw how to use the technology to send push notifications directly to a device based on the criteria set using Outlook Mobile Access. Mobile Information Server also allows you to browse your corporate intranet from a mobile device.

In the section on Outlook Mobile Manager, you installed the software, configured a profile, and saw how this rich client tool can help you make the best of the limited bandwidth and screen size of a mobile phone. These options included filtering and selecting the messages that you receive and compressing the size of your text messages. You saw how you can configure the software to send Outlook reminders just before those appointments and task deadlines. The system will even send you a summary of your meetings before the day begins.

In the section on SQL Server 2000 CE Edition, you saw that you can easily fetch relational data from a remote master database by connecting to a Web server. Through this connection, you can pull a very rich subset of corporate data and take it with you on your Pocket PC. Clearly, Microsoft has created technologies that enable Pocket PCs and mobile phones to access important information anytime, anywhere, and in whatever format the client supports.

Solutions Fast Track

Microsoft Mobile Information Server

☑ Connects to Exchange Server 2000 to give wireless devices access to Exchange inboxes.

☑ Supports push notifications of events by sending brief messages as e-mail directly to a mobile phone.

☑ Through Outlook Mobile Access, users can configure several options about the messages they receive on their mobile devices, filtering out all but the essential information.

Microsoft Outlook Mobile Manager

- ☑ A rich desktop environment for creating a custom profile that determines what messages appear on the mobile device and when you receive them.You can set up a schedule so that only urgent messages reach you after hours or on weekends.
- ☑ Features technologies that compress messages to save bandwidth on mobile phones. Compression can include automatic removal of spaces and vowels in the text.
- ☑ Lets you send an Outlook reminder to your mobile device.These can include notices of meetings and tasks and a summary of the next day's schedule.

Microsoft SQL Server 2000 CE Edition

- ☑ Provides a relational database on a small platform for the Windows CE platform such as the Pocket PC. SQL Server 2000 CE Edition supports the fundamental capabilities of the full SQL Server 2000 product.
- ☑ Lets you pull data from a master database and store it locally on the Pocket PC. On your device, you can view, sort, and update the information.
- ☑ Using merge replication, your application reconnects to the master SQL Server 2000 database and inserts your added records and updates. Although this is an advanced feature to develop and configure, it is easy for the end user.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: A great deal of software has been shown in this chapter. Where do I get it? Do I have to buy it just to try it?

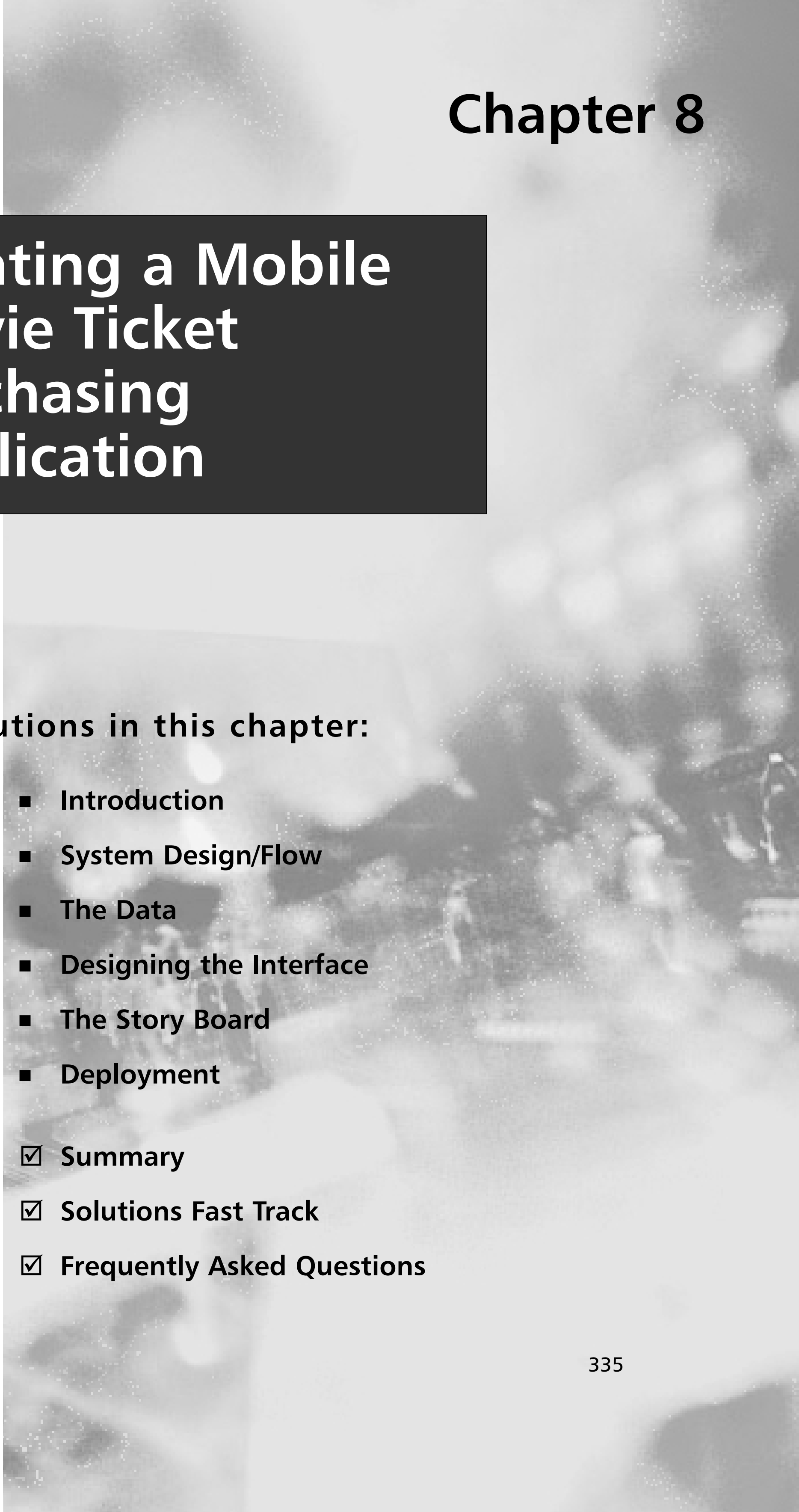
A: Probably the best way to obtain everything you need is by purchasing to the Microsoft Developer Network (MSDN) subscription CDs. MSDN sends monthly updates with the latest developer tools. Another way is to download evaluation versions of the tools from the Microsoft Download Center at www.microsoft.com/downloads/search.asp.

Q: What expertise is required for implementing and maintaining these solutions?

A: As you've seen, some of the installation and configuration is quite detailed and intricate. For a production environment, you should be an experienced Exchange 2000 administrator with a solid knowledge of Active Directory and Internet Information Server. That said, all of this is technology that you can learn from books like this.

Q: Do I need to buy a mobile phone and a Pocket PC to develop these technologies?

A: Strictly speaking, you can get by with the emulators that we've shown in this chapter. Emulators are handy tools, but they don't expose your solutions to the real world. To have confidence in the applications, you need to test your configurations on the targeted hardware—and the more variations of that hardware, the better.



Chapter 8

Creating a Mobile Movie Ticket Purchasing Application

Solutions in this chapter:

Introduction

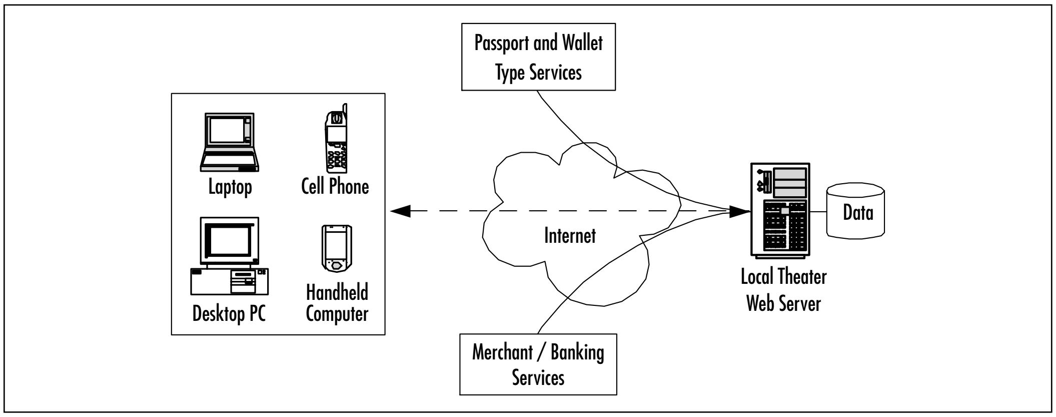
In this chapter, you will build a cinema ticketing system. This application will allow users to purchase theater tickets using mobile devices such as Web enabled cell phones and PDAs. The goal of this chapter is to show you how to use the .NET Mobile Framework to develop code that renders correctly in multiple mobile devices without actually programming using WML. As you build your application, you will test it using the following emulators (for more information on how to use and obtain these emulators, see Chapter 4):

- Mobile IE 3.0
- Pocket IE for the handheld PC
- Nokia 7110
- Openwave version 5.0
- Siemens S45

System Design/Flow

As you work through the process flow, you will notice subprocesses in the application that could have many different company- and vendor-specific implementations. In these areas, you can "stub out" interfaces and make assumptions about the return values. For simplicity of scope, this case study will involve a single fictitious cinema location called South Sound Cinemas. South Sound Cinemas has five screens and plays a different movie on each screen. Figure 8.1 shows the conceptual process overview.

Figure 8.1 Conceptual Overview



Basic Scenario

This section steps through the process of selecting and purchasing movie tickets online with a mobile device. This will give you an idea of the requirements of your hypothetical application:

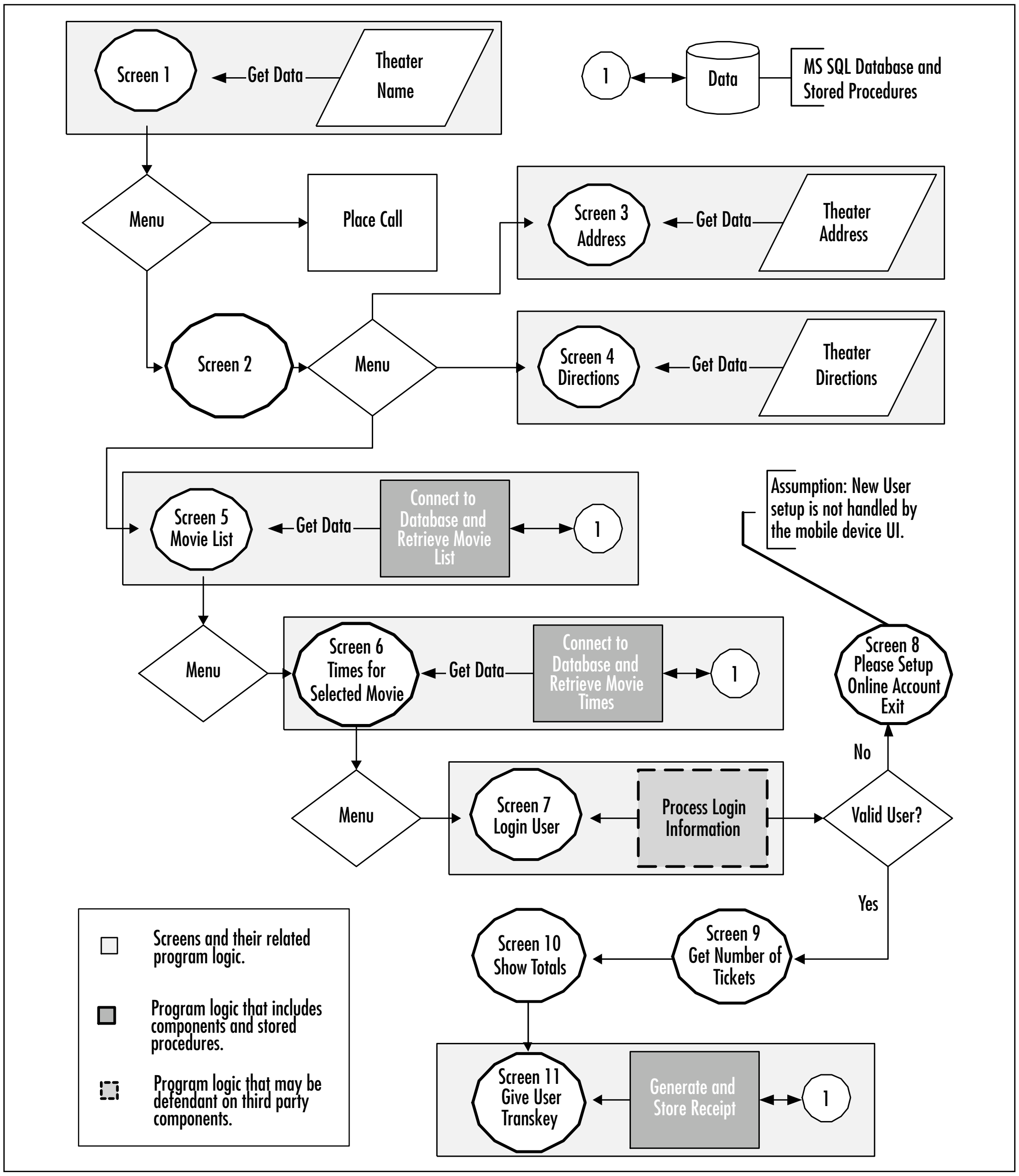
1. User accesses the theaters mobile site on the Web.
2. A list of theaters is displayed. From here, the user can also view more details about theater location (phone, address, and driving directions), as well as view a list of movies.
3. User selects a movie.
4. List of times the movie is showing is displayed.
5. User selects a showtime.
6. User selects the option to purchase ticket(s) (prompted for number of tickets).
7. Payment is processed and an authorization code is returned.
8. At the theater, the counter checks the computer system for authorization code from online/mobile purchases.

NOTE

A multitude of methods are used for online purchases—we do not go into the details here.

This can be broken down further by planning out the scenario as a flow chart (see Figure 8.2). This should give you a good idea of what you will need when designing your database schema. It also give you an idea of what types of data you will need to retrieve from the database, identifying candidates for stored procedures.

Figure 8.2 Process Flowchart


Screen 1
Get Data
Theater Name
1
Data
MS SQL Database and Stored Procedures
Menu
Place Call
Screen 3 Address
Get Data
Theater Address
Screen 2
Menu
Screen 4 Directions
Get Data
Theater Directions
Screen 5 Movie List
Get Data
Connect to Database and Retrieve Movie List
1
Assumption: New User setup is not handled by the mobile device UI.
Menu
Screen 6 Times for Selected Movie
Get Data
Connect to Database and Retrieve Movie Times
1
Screen 8 Please Setup Online Account Exit
No
Menu
Screen 7 Login User
Process Login Information
Valid User?
Yes
Screens and their related program logic.
Program logic that includes components and stored procedures.
Program logic that may be defendant on third party components.
Screen 10 Show Totals
Screen 9 Get Number of Tickets
Screen 11 Give User Transkey
Generate and Store Receipt
1

The Data

For this case study, you will be accessing data from your own database, from XML files on your server, and from a third-party vendor that provides electronic funds transfer services. This section focuses on the following:

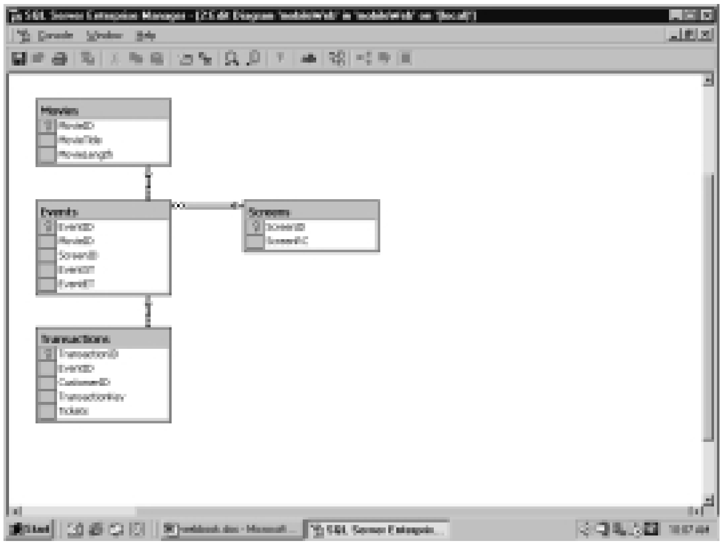
- **Database Design:**
  - Designing the database schema
  - Developing Stored procedures
  - Designing the DataAccess Component—a component wrapper around your stored procedures
- **Using XML as a data source** Developing a component interface for an XML datasource
- **Authenticating the user** Developing a component interface to simulate user login.

The following section takes an in-depth look at designing our database.

Database Design

For this case study, you will be using a SQL 2000 database. The database schema consists of four tables; you will exclude any customer information, because validation will be handled through a third party. Figure 8.3 shows the database diagram.

Figure 8.3 Database Diagram



The four tables consist of Movies, Events, Screens, and Transactions. In the transactions table, you will be simulating the return value for authentication purposes, either from Passport or a user's phone charge ID, into the *CustomerID* field column value. Each table is linked by their corresponding relationships. The Movies table will store the list of movies. The events table will store the separate times a movie plays. The screen table will list the screens available to the theater.

To interact with these tables, you will create three stored procedures. You will only be creating the application necessary tasks. You will not need to create administration functions, so you will not create administration-stored procedures. You do need to have a stored procedure return all movies from the movies table. Figure 8.4 shows that simple code.

Figure 8.4 Proc *returnAllMovies*

```
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS ON
GO



CREATE  PROCEDURE returnAllMovies
AS

SELECT * FROM Movies



GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO
```

This is as basic as it gets for a stored procedure. It is better to create stored procedures for all interaction with the database as opposed to using ad hoc queries in your source code—even if they are as rudimentary as this one. A stored procedure is precompiled code that will execute far faster than the ad hoc queries written inline.

Now that you have all the movies displayed via your stored procedure, you will need to get the time that those movies will be showing. You can accomplish this by another stored procedure that will take one input parameter, *movieID*, and return the specified times for that movie (see Figure 8.5).

Figure 8.5 Proc *getTimeofMovie*

```
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS ON
GO

CREATE PROCEDURE getTimeofMovie

           @MovieID int

AS

SELECT e.EventST, e.EventID
FROM Events e
JOIN Movies m ON e.MovieID = m.MovieID
WHERE m.MovieID = @MovieID

GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO
```

Here you supply the input parameter *@MovieID*, and you will do a simple inner join to get to the corresponding times based on the event.

Now the final step is to actually create the transaction to purchase a movie ticket. You can do this with yet another stored procedure. Figure 8.6 shows the code to do this. Remember that you are going to be self-generating your Customer ID field.

Figure 8.6 Proc *buyTickets*

```
CREATE    PROC buyTickets

        @EventID int,
        @CustomerId int,
        @Tickets int
AS
          Declare @transKey nvarchar(50)
          SELECT @transKey = Cast(((((@EventID + @CustomerId)* 5) *
              10) / 2) as nvarchar) + '.' +
          CAST(@EventID as nvarchar)+ '.' + CAST(@CustomerId as
              nvarchar)

        DECLARE @TotalSpace nvarchar(50)

        SET @TotalSpace = (SELECT s.ScreenRC FROM Screens s
                   JOIN Events e on e.ScreenID = s.ScreenID
                   WHERE e.EventID =  @EventID)

        DECLARE @TakenSpace nvarchar(50)

        SET @TakenSpace = (SELECT SUM(Tickets)
                   FROM Transactions
                   WHERE EventID = @EventID)

BEGIN TRANSACTION purchase

IF (@Tickets + CAST(@TakenSpace AS int)) > CAST(@TotalSpace AS int)

        BEGIN
        ROLLBACK TRAN purchase
        RETURN 0
        END
ELSE
```

Continued

Figure 8.6 Continued

```
    INSERT INTO Transactions
    (EventID, CustomerID, TransactionKey, Tickets)VALUES
    (@EventID,@CustomerId,@transKey,@Tickets)

COMMIT TRANSACTION purchase

    Select @transKey

GO
```

The stored procedure *buyTickets* takes in three parameters, *@EventID*, *@CustomerID*, and *@Tickets*. The first thing you do in this proc is to generate a unique transaction key to pass to the event for verification at the movie theater. Let's look at that in more detail:

```
Declare @transKey nvarchar(50)
SELECT @transKey = Cast(((((@EventID + @CustomerId)* 5) * 10) / 2) as
    nvarchar) + '.' + CAST
    (@EventID as nvarchar)+ '.' + CAST(@CustomerId as nvarchar)
```

First you declare a local variable and then use a combination of string concatenations and mathematical operations to generate a unique number based on the event and the customer.

Next you need to find out what space is available for a specific event and hold that data in a local variable also. You do that with the following code:

```
DECLARE @TotalSpace nvarchar(50)

        SET @TotalSpace = (SELECT s.ScreenRC FROM Screens s
                    JOIN Events e on e.ScreenID = s.ScreenID
                    WHERE e.EventID =  @EventID)
```

Here you set another local variable equal to the result of a specific inner join off the Events table to the Screens table to get at the room capacity. Now you need to know how much space has been taken up already—you do not want to oversell a particular venue. You can achieve this by doing the following:

```
DECLARE @TakenSpace nvarchar(50)

        SET @TakenSpace = (SELECT SUM(Tickets)
                   FROM Transactions
                   WHERE EventID = @EventID)
```

Again you declare a local variable and set it equal to a result set of tickets sold for a specific event. Now you can do a simple *if* statement to see if the tickets you want to purchase plus the tickets already purchased are greater than the space available:

```
IF (@Tickets + CAST(@TakenSpace AS int)) > CAST(@TotalSpace AS int)
```

And finally you want to create a transaction that holds this, and if it fails, roll back and do not insert the new row:

```
 BEGIN TRANSACTION purchase

IF (@Tickets + CAST(@TakenSpace AS int)) > CAST(@TotalSpace AS int)

        BEGIN
        ROLLBACK TRAN purchase
        RETURN 0
        END
ELSE

    INSERT INTO Transactions
    (EventID, CustomerID, TransactionKey, Tickets)VALUES
    (@EventID,@CustomerId,@transKey,@Tickets)


COMMIT TRANSACTION purchase

    Select @transKey

GO
```

The final piece is the return of the variable *@transKey*. The person to verify that they have purchased a ticket for the movie will use this. This is all you need on the database end for this application. The next section moves on to the data access component that will use the stored procedures you just created.

Designing the *DataAccess* Component

In the previous section, you created a simple database and stored procedures to handle retrieving a list of movies (*returnAllMovies*), retrieving a list of showtimes for a given movie (*getTimeofMovie*), and processing a ticket purchase (*buyTickets*).

In this section, you will be creating one component class to handle all data access with your MS SQL Server 2000 database. This component will provide method wrappers for your stored procedures. They will be used to connect to the database and execute each of these procedures. This will help to illustrate not only what is going on in the process, but also show the component model .NET has built in for MS SQL Server. You can find this file at **www.syngress.com/solutions** in the components folder of the project (see dataaccess.vb).

Figure 8.7 shows the code for this component, and the remainder of this section breaks it down piece by piece. This component will handle all data access for your application. You will use this component to interact with the SQL Server 2000 database and execute all of your stored procedures. Having the code separated in this manner will be easier to maintain and scale later.

Figure 8.7 Dataaccess.vb

```
Imports System.Data
Imports System.Data.SqlClient

Public Class dataAccess

Private conStr As String = "server=(local); database=mobileWeb;" _
                           & "Trusted_Connection=yes" 'default value

    Private Conn As SqlConnection = New SqlConnection()
    Private Cmd As SqlCommand = New SqlCommand()
    Private ds As DataSet = New DataSet()

    Public Property ConnectionString() As String
```

Continued

Figure 8.7 Continued

```
        Get
            Return conStr
        End Get
        Set(ByVal Value As String)
            conStr = Value
        End Set
    End Property

    Private Sub initializeProcedure(ByVal CommandText As String)
        Conn.ConnectionString = conStr
        Cmd.Connection = Conn
        Cmd.CommandType = CommandType.StoredProcedure
        Cmd.CommandText = CommandText
    End Sub

    Public Function loadMovies() As DataSet
        initializeProcedure("returnAllMovies")
        Dim da As SqlDataAdapter = New SqlDataAdapter(Cmd)
        Conn.Open()
        da.Fill(ds, "movie")
        Conn.Close()
        loadMovies = ds
    End Function

    Public Function getTimes(ByVal MovieId As Integer) As DataSet
        initializeProcedure("getTimeofMovie")
        Cmd.Parameters.Add( _
New SqlParameter("@MovieID", SqlDbType.Int, 4 ))
        Cmd.Parameters("@MovieID").Value = MovieId
        Dim da As SqlDataAdapter = New SqlDataAdapter(Cmd)
        Conn.Open()
        da.Fill(ds, "event")
        Conn.Close()
```

Continued

Figure 8.7 Continued

```
        getTimes = ds
    End Function

    Public Function buyTickets( _
           ByVal EventID As Integer, _
           ByVal CustomerID As String, _
           ByVal Tickets As Integer) As String
        initializeProcedure("buyTickets")
        Cmd.Parameters.Add( _
           New SqlParameter("@EventID", SqlDbType.Int, 4 ))
        Cmd.Parameters("@EventID").Value = EventID

        Cmd.Parameters.Add( _
           New SqlParameter("@CustomerID", SqlDbType.Int, 4))
        Cmd.Parameters("@CustomerID").Value = CustomerID
        Cmd.Parameters.Add( _
           New SqlParameter("@Tickets", SqlDbType.Int, 4 ))
        Cmd.Parameters("@Tickets").Value = Tickets

        Dim results As String
        Cmd.Connection.Open()
        Dim sdr As SqlDataReader =
    Cmd.ExecuteReader(CommandBehavior.CloseConnection)
        While sdr.Read()
            results = sdr.GetString(0)
        End While
        sdr.Close()
        Conn.Close()
        If Len(results) > 0 Then
            buyTickets = results
        Else
            buyTickets = "Transaction not processed"
        End If
```

Continued

Figure 8.7 Continued

```
    End Function

End Class
```

You first need a set of private variables that you will use for all the functions in the class. You will also create a function that will load and initialize the command text so that it too will need to be coded only once:

```
Private conStr As String = "server=(local); database=mobileWeb;" _
                           & "Trusted_Connection=yes" 'default value

    Private Conn As SqlConnection = New SqlConnection()
    Private Cmd As SqlCommand = New SqlCommand()
    Private ds As DataSet = New DataSet()

    Public Property ConnectionString() As String
        Get
            Return conStr
        End Get
        Set(ByVal Value As String)
            conStr = Value
        End Set
    End Property

    Private Sub initializeProcedure(ByVal CommandText As String)
        Conn.ConnectionString = conStr
        Cmd.Connection = Conn
        Cmd.CommandType = CommandType.StoredProcedure
        Cmd.CommandText = CommandText
    End Sub
```

The first function is *loadMovies()*. This will make a call to the stored procedure (proc) that you created in the database and load the data into a *DataSet* via your *SQLDataAdapter*:

```
    Public Function loadMovies() As DataSet
```

```
        initializeProcedure("returnAllMovies")
        Dim da As SqlDataAdapter = New SqlDataAdapter(Cmd)
        Conn.Open()
        da.Fill(ds, "movie")
        Conn.Close()
        loadMovies = ds
    End Function
```

Once your resultset is in a *DataSet*, you can manipulate that data for display in a number of ways. You will be focusing on the XML object, but there are many controls and objects that can render the data from this point.

The second function in your class will return the times for a specific movie. Again you will access the proc that you created to handle this and load the result set into the *DataSet*:

```
Public Function getTimes(ByVal MovieId As Integer) As DataSet
        initializeProcedure("getTimeofMovie")
        Cmd.Parameters.Add( _
            New SqlParameter("@MovieID", SqlDbType.Int, 4))
        Cmd.Parameters("@MovieID").Value = MovieId

        Dim da As SqlDataAdapter = New SqlDataAdapter(Cmd)
        Conn.Open()
        da.Fill(ds, "event")
        Conn.Close()
        getTimes = ds
    End Function
```

Just as the first function used the *SQLDataAdapter* to load the result set into the *DataSet*, so does this one. Remember that this function returns a *DataSet*. The next piece of the class is the *buyTickets()* function.

For this function, you will pass in the three input parameters required by the proc that you already created. You will check to see if the stored procedure returns the *@TransKey* value, which will indicate that the transaction in your stored procedure had gone through successfully:

```
  Public Function buyTickets(ByVal EventID As Integer, _
          ByVal CustomerID As String, _
```

```
        ByVal Tickets As Integer) As String
      initializeProcedure("buyTickets")
      Cmd.Parameters.Add( _
        New SqlParameter("@EventID", SqlDbType.Int, 4))
      Cmd.Parameters("@EventID").Value = EventID

      Cmd.Parameters.Add( _
        New SqlParameter("@CustomerID", SqlDbType.Int, 4))
      Cmd.Parameters("@CustomerID").Value = CustomerID

      Cmd.Parameters.Add( _
        New SqlParameter("@Tickets", SqlDbType.Int, 4))

      Cmd.Parameters("@Tickets").Value = Tickets

      Dim results As String
      Cmd.Connection.Open()
      Dim sdr As SqlDataReader =
           Cmd.ExecuteReader(CommandBehavior.CloseConnection)
      While sdr.Read()
          results = sdr.GetString(0)
      End While
      sdr.Close()
      Conn.Close()
      If Len(results) > 0 Then
          buyTickets = results
      Else
          buyTickets = "Transaction not processed"
      End If
  End Function
```

The next section covers designing the user interface (UI).You will be using this component to retrieve and send data between your UI components and your database.

Using XML as a Data Source

For this application, you will store information about the theater in an XML file, TheaterInfor.xml (see Figure 8.8). In this application, the theater information is relatively static. Rather than access the database for this information each time the site is accessed, you can simply store the data in an XML file on the server. The *LoadTheaterDetails* component will provide a component interface to the XML data file (note that you can extend this component to update your XML data source if you want to provide a nice interface for ZIP code, area code, or hours of operation changes, and so on.) In your mobile Webforms, you will use this component to retrieve data from this XML file. You can find this file at **www.syngress.com/solutions** (see TheaterInfo.xml).

Figure 8.8 Theater Description File (TheaterInfo.xml)

```
<?xml version="1.0" encoding="utf-8" ?>
<theaters>
  <theater id="1">
    <name>South Sound Cinemas</name>
    <hours>5:00 pm - Midnight</hours>
    <phone>(123) 456-7890</phone>
    <street>123 S. Sound Blvd.</street>
    <city>South Sound</city>
    <state>WA</state>
    <zip>98765-4321</zip>
    <directions>
      from I-5 south, take exit 189B, turn left at 5th, at the
      next stop light,  turn right onto S. Sound Blvd, we are located on
      the right-hand side of the street.
    </directions>
    <map>map.gif</map>
  </theater>
</theaters>
```

In the next section, you will build a component that will retrieve data from the XML file.

SYNGRESS syngress.com

Creating the *LoadTheaterDetails* Component

The *LoadTheaterDetails* component is designed to retrieve data from the XML data file. You can find it at **www.syngress.com/solutions** (see Load-TheaterDetails.vb in the components folder of the Web Application). This section examines the code for the component piece-by-piece. Because you will be retrieving data from an XML file, you will need to import support classes for XML and Xpath. You will need to add the following to the top of your class page:

```
Imports System
Imports System.Xml
Imports System.Xml.XPath
```

Also, because you will be loading an XML file from the server virtual root, you will need to use the *MapPath* property of the Server object. Support for the Server object is included in the *System.Web.UI.Page* namespace. To use the Server object in your class, your class will need to inherit this namespace. You can accomplish this by adding the *inherits* declaration within your class definition:

```
Public Class loadTheaterDetails
    Inherits System.Web.UI.Page  'needed for "Server" object
```

Next, declare private variables and a property that allows you to set the path to your XML file:

```
    Private path As String = "TheaterInfo.xml"
    Private doc As New XmlDocument()
    Private count As Int32

    Public Property FilePath() As String
        Get
            Return path
        End Get
        Set(ByVal Value As String)
            path = Value
        End Set
    End Property
```

The class must be able to load your XML file into a Document object:

```
    Public Function loadDetails() As String
```

```
        Try
            doc.Load(Server.MapPath(path))
            loadDetails = "success"
            count = doc.SelectNodes("//theater").Count
        Catch e As Exception
            loadDetails = "An error occured: " + e.Message
        End Try
    End Function
```

Next you must define methods that return data from element nodes. Because your XML data file is rather flat, you can easily use this first function to retrieve data from almost all of your nodes:

```
   Public Function getTextNode(
                   ByVal id As Int32,
                   ByVal nodename As String) As String
       Dim xpath As String = "//theater[ @id='" & id & "']/" &
           nodename
       getTextNode = doc.SelectSingleNode(xpath).InnerText
   End Function
```

This function will enable you to retrieve data from any node by using a valid Xpath query:

```
   Public Function getTextNodeWithQuery(
                   ByVal XpathQuery As String) As String
      getTextNodeWithQuery = doc.SelectSingleNode(XpathQuery).InnerText
   End Function
End Class
```

This component fulfills your requirement of being able to retrieve data from your XML file. The next section looks at building yet another component that you will use to simulate user login.

Authenticating the User

You can handle user authentication in several different ways. You could create your own signup page that would require the user to supply information that you would store in your database; you could already have a member service program in place that you could tap into; or you could use a third-party system such as

Microsoft Passport. No matter which method you employ, basically you would programmatically call a function that would return a user ID if the user were authenticated. So, you will create a component with a method that simulates authenticating a user and returns a user ID.

Creating the *mockAuthenticate* Component

The *mockAuthenticate* component is designed to simulate a user login; it returns a mock user id. You can find it at **www.syngress.com/solutions** (see mockAuthenticate.vb in the components folder of the Web Application). Figure 8.9 shows the code for the component.

Figure 8.9 mockAuthenticate.vb

```
Imports System

Public Class mockLogin
   Public Function authenticateUser(
                   ByVal login As String,
                   ByVal password As String) As String
       'login authentication logic or method call could go here.
       'could use a standard DB, possibly passport or a similar
       'service
       'this is some dummy logic to generate a customerID
```

Create a mock id that appears somewhat unique:

```
       Dim customerid As String
       Dim temp As Array = login.Split(".")
       Dim logLen As Int32 = Len(login)
       Dim passLen As Int32 = Len(password)
       customerid = logLen & passLen & (passLen * passLen)
       authenticateUser = customerid
   End Function
End Class
```

In this section ("The Data"), you have created the back-end and middle-tier structure you need to support your application. The next section ("Designing the

Interface”) focuses on the front-end and the pieces that connect your UI to your middle-tier and back-end processes.

Developing & Deploying…

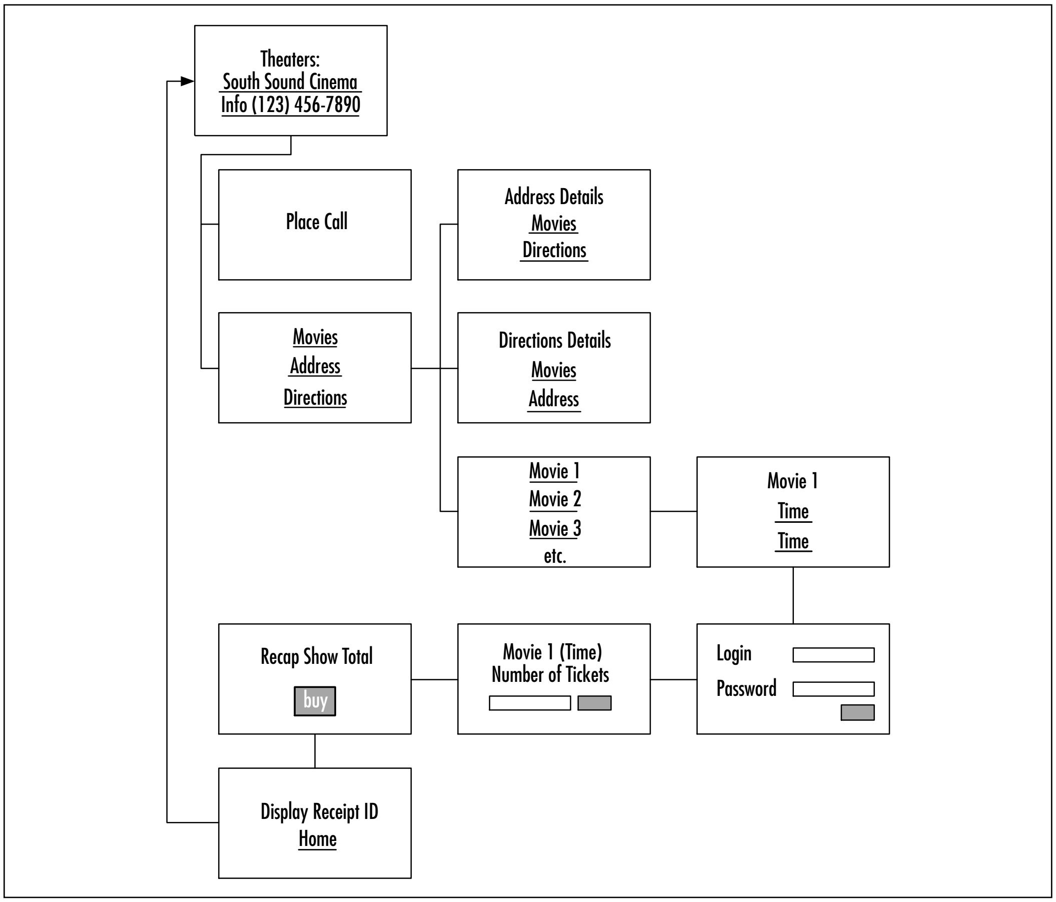
Using the Right Mobile Web Form Control for Device Compatibility

Although each control offers different features, not all controls will render correctly on all devices. For instance, the *ObjectList* control is very powerful and allows for diverse and smoothly transitioning UI for the mobile phone emulators we’ve tested with, however, some of its features are inaccessible when testing against the Pocket IE emulator. Also Pocket IE is unable to handle multiple form controls. This is common for creating multiple WML cards commonly used to improve the user experience for cellular phones. These shortcomings may be due to the device-specific rendering built into this beta version of the mobile framework. Bottom line: If you need to support both Web-enabled phones and handheld CE devices, keep it simple.

Designing the Interface

The interface will be straightforward and simple. Mobile .NET Web Forms are capable of rendering differently for various devices, using default or custom style properties. For simplicity, you will work with only a few styles. Feel free to adjust fonts and background colors, because these settings will be rendered only by devices that support them, but will not cause devices that don’t to fail. In this section, you will be creating the front end and the logic that ties your UI to your data sources. Figure 8.10 shows the pages that make up the UI (see Figure 8.10).

Figure 8.10 UI Storyboard



Developing & Deploying…

Mobile: Label Control versus Mobile: TextView Control

The *Mobile:Label* and *Mobile:TextView* controls are very similar in functionality. An important difference is that the *Label* control will display only text; the *TextView* control will actually render any markup that is included with the text.

The *TextView* control allows you to add some WML to the text rendered, whereas *Label* does not. For example, look at the following string:

Continued

```
"Buy tickets for:<br /><b>title</b>"
```

Label renders it as follows:

```
Buy tickets for:<br /><b>title</b>
```

TextView renders it like this:

```
Buy tickets for:

title
```

Note that *Label* escapes the <br /> tag and renders it as *
*.

Creating the Home page: Default.aspx

You can find the files for this page at **www.syngress.com/solutions** (see Default.aspx and Default.aspx.vb). Figure 8.11 shows the code for the first page that the user will navigate to, therefore it needs to display three things:

- Caption: theater
- Theater name
- Phone number link

Here's how to create it step-by-step:

1. Add a new Mobile WebForm to your project, and name it **default.asp**. This will be the home page.
2. In design view, drag a *TextView*, a *Link*, and a *Call*, control onto the form.
3. Now look at the code in HTML view.
4. Add the attribute *Font-Bold* and set it to true in the *TextView* control.
5. Add the attribute *NavigateURL* and set it to "menu1.aspx". (This will be the next page.)

Your code should look like what is shown in Figure 8.11.

Figure 8.11 Default.aspx

```
<body xmlns:mobile="Mobile Web Form Controls">
  <mobile:form id="Form1" runat="server">
    <mobile:TextView id="Title" runat="server" Font-Bold="True" />
    <mobile:Link id="theater" runat="server" NavigateURL="menu1.aspx" />
    <mobile:Call id="callTheater" runat="server"></mobile:Call>
  </mobile:form>
</body>
```

In the code-behind page (default.aspx.vb), add a reference to the *LoadTheaterDetails* component:

```
Imports MobileCinema.LoadTheaterDetails
```

In the *page_load* method, create an instance of *LoadTheaterDetails* and call the *load* method:

```
Dim xData As New LoadTheaterDetails()
Dim result As String = xData.loadDetails()
```

Set the value of the Link control to the name element of your XML document:

```
theater.Text = xData.getTextNode(1,"name")
```

The Call control enables phones that support it to create a "call now" link on the page. Mobile devices that don't support it will simply render the text. You will need to set the phone number value to the value of the phone element in your XML document:

```
callTheater.Text = "Info Line"
callTheater.PhoneNumber = xData. getTextNode(1,"phone")
```

Next you need to set the value of the caption. Because some mobile browsers will put the link button at the top with the next element to the right of it, you will need to add a "<br>" tag to force a new line (see Figures 8.12 and 8.13):

```
Title.Text = "<br />Theater"
```

For Nokia, Pocket IE, the Siemens S45, and Openwave emulators, the layout is not affected (see Figures 8.13 through 8.16).

Figure 8.12 Default Mobile Explorer (
 Is Needed)

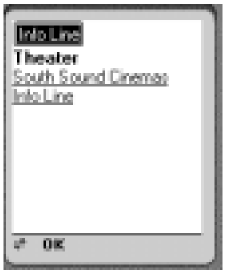


Figure 8.13 Siemens S45 (
 Is Irrelevant)

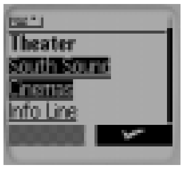


Figure 8.14 Openwave 5 (
 Is Irrelevant)

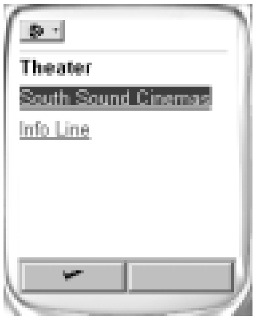


Figure 8.15 Nokia 7110 (
 Is Irrelevant)

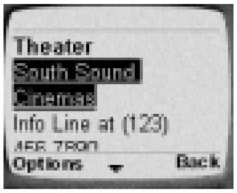
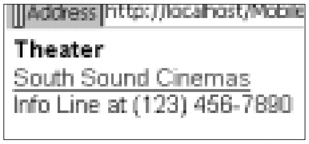


Figure 8.16 PocketIE (
 Is Irrelevant)



Creating Menu, Directions, and the Details Page

The menu, directions, and details pages are very similar to the default page. You can view their source code at **www.syngress.com/solutions** (see menu.aspx, menu.aspx.vb, directions.aspx, directions.aspx.vb, details.aspx, and details.aspx.vb in the project folder).

Creating the Movie List Page: MovieList.aspx

You can find the files for the movie list page at **www.syngress.com/solutions** (see MovieList.aspx and MovieList.aspx.vb). The code shown in Figure 8.17 uses only one list control to display a list of movies from the database. You need each movie not only to be listed but also to be a link so that the user can drill down. You can accomplish this by simply adding the attribute *ItemsAsLinks* and set it to true.

Figure 8.17 MovieList.aspx

```
<body xmlns:mobile="Mobile Web Form Controls">
  <mobile:Form id="Form1" runat="server">
    <mobile:List id="movieList" runat="server" ItemsAsLinks="True" />
  </mobile:Form>
</body>
```

Now let's look at the code-behind page for Figure 8.18 (MovieList.aspx.vb) to see how you access the database. In your page class, you will need to add a reference to and create an instance of the *dataAccess* component.

Figure 8.18 MovieList.aspx.vb

```
Public dso As dataAccess = New dataAccess()
```

You will be accessing the *loadMovies* method of the *dataAccess* component. This method returns a *DataSet*. So, you must first create a variable of type *DataSet*:

```
Public DS As DataSet
```

In the *page_Load* method, you will bind your *DataSet* to the list control:

```
DS = dso.loadMovies()
movieList.DataSource = DS.Tables(0)
```

Next you will assign the displayed text to the *movieTitle* column of your datasource:

```
movieList.DataTextField = "MovieTitle"
```

This will write the *MovieTitle* field values to the text portion of an anchor tag or its WML equivalent (depending on the device). Next you will assign a reference to the movie to the link field (the *href* value of an anchor tag):

```
movieList.DataValueField = "MovieID"
```

The anchor tag generated would look similar to this:

```
<a href= "1" >Cast Away</a>
```

This makes the link pretty useless, but you do need the *movieId* in the link. To fix this, you need to loop through the collection after it is generated and update the value:

```
Dim i As Int32
Dim title As String
For i = 0 To movieList.Items.Count - 1
   title = HttpUtility.UrlEncode(movieList.Items(i).Text)
   movieList.Items(i).Value = "showtimes.aspx?movieid=" _
               & movieList.Items(i).Value & "&title=" & title
Next
```

Now a generated anchor tag will look like this:

```
<a href="showtimes.aspx?movieid=1&title= Cast%20Away" >Cast Away</a>
```

Note the use of *HttpUtility.UrlEncode*. This ensures the link is encoded properly for http and that all special characters are converted to their ASCII equivalent (notice in the link that *UrlEncode* replaces the space character with *%20*.) In order to use this method, you must be sure to include the following reference at the top of the page.

```
Imports System.Web ' needed for UrlEncode
```

Creating the Showtimes: showTimes.aspx

The page showTimes.aspx, which list the times for a given movie, is nearly identical in concept to the MovieList page. You can view the source code for it at **www.syngress.com/solutions** (see showTimes.aspx and showTimes.aspx.vb).

Creating the Login Page: purchase1.aspx

You can find the files for the login page at **www.syngress.com/solutions** (see purchase1.aspx and purchase1.aspx.vb). When a user has selected a movie and a showtime and is ready to make a purchase, you must first authenticate them. Authentication could be with your own database of members, or possibly a vendor service such as Passport (see the discussion on this topic earlier in this chapter in the section "Authenticating the User").

On this page, you will provide an interface that allows the user to enter a username and password to submit for authentication. (Note: Because this example uses a mock authentication, any username and password will be allowed.) In Figure 8.19, note the use of the mobile form validation controls *RequiredFieldValidator* and *RegularExpressionValidator*. These are much like their standard Web Form counterparts, however, none of them appear to function with any of our emulators, perhaps this will be fixed by the time VSNET is released.

Figure 8.19 purchase1.aspx

```
<mobile:form id="Form1" runat="server">
   <mobile:TextView id="TextView1" runat="server"></mobile:TextView>
Create a label and textbox for the user to enter their login
   <mobile:Label id="Label1" runat="server">Login</mobile:Label>
   <mobile:TextBox id="login" runat="server"></mobile:TextBox>
```

Validation controls for the login field are designed to ensure that the field is not empty and that it conforms to the minimum requirements of an e-mail address.

```
   <mobile:RequiredFieldValidator id="RequiredFieldValidator1"
            runat="server"
            ControlToValidate="login"
            ErrorMessage="*">
   </mobile:RequiredFieldValidator>
   <mobile:RegularExpressionValidator id="RegularExpressionValidator1"
            runat="server"
            ControlToValidate="login"
            ErrorMessage="RegularExpressionValidator"
            ValidationExpression="^.*@.*\..*">
```

```
</mobile:RegularExpressionValidator>
```

Create a label and textbox for the user to enter her password:

```
<mobile:Label id="Label2" runat="server">Password</mobile:Label>
<mobile:TextBox id="password" runat="server" Password="True">
</mobile:TextBox>
```

Validation control for the password field is designed to ensure that the field is not empty.

```
<mobile:RequiredFieldValidator id="RequiredFieldValidator2"
        runat="server"
        ControlToValidate="login"
        ErrorMessage="*">
</mobile:RequiredFieldValidator>
```

Add a Submit button to pass the data to the server for processing:

```
   <mobile:Command id="Command1" runat="server">Submit</mobile:Command>
</mobile:form>
```

Look at the code-behind page shown in Figure 8.20 (purchase1.aspx.vb). The first time the page is loaded, you want to store the data passed in the querystring from the previous page into *Session* variables so that you can use their values in upcoming pages.

Figure 8.20 purchase1.aspx.vb

```
If Not IsPostBack Then
    Session("eventID") = Request.QueryString("eventid")
    Session("movietitle") =
      HttpUtility.UrlDecode(Request.QueryString("title"))
    Session("starttime") = Request.QueryString("time")
End If
```

Next you want to add code to the click event of the Submit button. First you want to call the *authenticateUser* method of your *dataAccess* component and pass it the values of login and password. You will store this mock userID in a *Session* variable so that you can use its value in upcoming pages:

```
Session("customerID") =
   logUser.authenticateUser(login.Text, password.Text)
```

Now all that is left to do is take the user to the next screen:

```
Response.Redirect("purchase2.aspx")
```

The following page will handle prompting the user for the number of tickets they would like to purchase.

Creating the Tickets Page: purchase2.aspx

You can find the files for the tickets page at **www.syngress.com/solutions** (see purchase2.aspx and purchase2.aspx.vb). This page is relatively simple; it contains a *TextView* control that displays the movie and time selected, some static text "# of tickets," a textbox for user entry, and a Submit button. Figure 8.21 shows the code for purchase2.aspx.

Figure 8.21 purchase2.aspx

```
<mobile:Form id="Form1" runat="server">
   <mobile:TextView id="recap" runat="server"></mobile:TextView>
   <mobile:TextView id="TextView1" runat="server">
      # of Tickets
   </mobile:TextView>
   <mobile:TextBox id="ticketCount" runat="server"></mobile:TextBox>
   <mobile:Command id="Command1" runat="server">submit</mobile:Command>
</mobile:Form>
```

The code-behind page for purchase2.aspx.vb (as seen in Figure 8.22) is also relatively simple. In the *page_load* method, you simply retrieve the values for the movie title and showtime the user selected earlier.

Figure 8.22 purchase2.aspx.vb

```
Dim movietitle As String = Session("movietitle")
Dim starttime As String = Session("starttime")
```

Now set it to the *Text* property of your *TextView* control "recap":

```
recap.Text = "<b>" & movietitle & "</b> [" & starttime & "]"
```

Next you need to add some code to the *click* event of the Submit button. Here is where you add some interesting code. The Openwave 5 emulator has a problem with appending letters to the value of this text field (interestingly, this is not a problem with Openwave 4.1; it also happens only when using cookieless sessions—required by the Nokia emulator). This shows up as an invalid cast exception on the next page when you try to use the value as an integer, the following is a workaround.

Store the value of the Textbox in a variable:

```
Dim result As String = ticketCount.Text
```

Search the variable for any occurrence of a letter. If a letter is found, store its char position:

```
Dim index As Int32 =
result.IndexOfAny("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
     )
```

If a letter is found, *index* will hold its position and set *result* to all the characters up to the first occurrence of a letter:

```
If index > 0 Then
   result = result.Substring(0, index)
End If
```

Next store the numeric result in a *Session* variable for use in upcoming pages:

```
 Session("count") = result
```

Now all that is left is to send the user to the next page:

```
 Response.Redirect("recap.aspx")
```

For this page, you can handle the user input of the number of tickets they would like. You can also examine a workaround for the Openwave 5 emulator. In the next section, you will display the details of the purchase.

SYNGRESS
syngress.com

Creating the Recap Page: recap.aspx

You can find the files for the recap page at **www.syngress.com/solutions** (see recap.aspx and recap.aspx.vb). So far, the user has selected a movie, a showtime, and the number of tickets they would like. On this page, you will display this information for the user along with the purchase price details. You will then enable them to approve processing the purchase:

```
<mobile:Form id="Form1" runat="server">
```

To display what the user has selected, use a *TextView* control:

```
<mobile:TextView id="recap"
   runat="server">TextView</mobile:TextView>
```

Next display the cost breakdown for the purchase:

```
<mobile:Label id="subValue" runat="server" Alignment="Right">
</mobile:Label>
<mobile:Label id="TaxValue" runat="server" Alignment="Right">
</mobile:Label>
<mobile:Label id="totalValue" runat="server" Alignment="Right">
</mobile:Label>
```

Lastly enable them to select the Buy link:

```
   <mobile:Link id="Link1" runat="server" NavigateURL="purchase3.aspx">
      buy
   </mobile:Link>
</mobile:Form>
```

Now let's examine the code-behind for this page (recap.aspx.vb). To simplify things, the price of the tickets is hardcoded, as is the local excise tax:

```
Public ticketprice As Double = 7.25
Public taxRate As Double = 0.08
```

In the *page_load* method, first retrieve the ticket information the user has provided:

```
Dim movietitle As String = Session("movietitle")
Dim starttime As String = Session("starttime")
Dim count As Int32 = CType(Session("count"), Int32)
```

Next calculate the total ticket price and the total price plus sales tax:

```
Dim subtotal As Double = count * ticketprice
Dim total As Double = subtotal * (1 + taxRate)
```

Now display the ticket information:

```
recap.Text = count & " for <br /><b>" & movietitle & "</b> [" &
starttime & "]<br />"
```

Here you display the price for the tickets, the tax amount, and the total with tax:

```
subValue.Text = FormatNumber(subtotal, 2) & " sub "
TaxValue.Text = FormatNumber(taxRate * subtotal, 2) & "  tax "
totalValue.Text = "$ " & FormatNumber(total, 2) & "  tot "
```

NOTE

Use the VB *FormatNumber* function to ensure that your numbers have two decimal places.

Now that the user has a breakdown of the cost, they can use the Back button on the mobile device to make changes, or they can select the Buy link, which will take them to the next page, which handles the processing of the order (purchase3.aspx).

Creating the Purchase Page: purchase3.aspx

You can find the files for the purchase page at **www.syngress.com/solutions** (see purchase3.aspx, and purchase3.aspx.vb). Earlier in the chapter, we discussed how to simulate the login process; similarly, you can also make this generalization with regards to purchase. You can handle collecting of funds in a multitude of ways: third-party merchant services, in-house processing with appropriate infrastructure, online Wallets, even billing the cell phone account directly. (Nokia offers this on some models, mobile purchases are reflected on the cell phone monthly bill.)

Your application assumes that this part is taken care of and generate a "Transaction key" to simulate the completion of the transaction. The Transaction key is what is used to grant theater access to the user. The Logic for this is encapsulated in the *dataAccess* component's *buyTickets* method and the *buyTickets* stored procedure. Figure 8.23 consists of several text controls and a link to the home page.

SYNGRESS syngress.com

Figure 8.23 purchase3.aspx

```
<mobile:Form id="Form1" runat="server">
   <mobile:TextView id="TextView1" runat="server"></mobile:TextView>
   <mobile:TextView id="TextView2" runat="server"></mobile:TextView>
   <mobile:TextView id="TextView3" runat="server"></mobile:TextView>
```

Continued

Figure 8.23 Continued

```
   <mobile:Link id="home" runat="server" NavigateURL="default.aspx">
      home
   </mobile:Link>
</mobile:Form>
```

Figure 8.24 shows the code-behind for purchase3.aspx.vb:

Figure 8.24 purchase3.aspx.vb

```
Private Sub Page_Load(
                  ByVal sender As System.Object,
                  ByVal e As System.EventArgs) Handles MyBase.Load
```

Get transaction information stored in session variables from earlier steps in the process (pages):

```
   Dim eventid As Int32 = CType(Session("eventid"), Int32)
   Dim customerid As Int32 = CType(Session("customerid"), Int32)
   Dim count As Int32 = CType(Session("count"), Int32)
```

Call the *buyTickets* method of your *dataAccess* component (note: *dso* is a local instance of the *dataAccess* component.) This is your stub for a third-party vendor electronic funds transfer component tie-in. It will simulate billing the customer and will return a mock authorization key:

```
   Dim transkey As String = dso.buyTickets(eventid, customerid, count)
   TextView1.Text = "Thank you for your purchase!" & "<br />
   your transaction key is:<br />"
   TextView2.Font.Bold = BooleanOption.True
   TextView2.Font.Name = "arial"
   TextView2.Text = transkey
   TextView3.Text = "bookmark this page."
End Sub
```

This page will display a message including the newly generated transaction key. It will also display the message, "bookmark this page." For a mobile phone, bookmarking the page will store it and enable it to easily be retrieved later. The following section examines how each of the emulators displays your mobile application.

The Story Board

This section contains a walkthrough of the application UI for each of the emulators we tested with. For the most part, the UI is the same. However, some of the smaller interfaces create more need to scroll. Interestingly, the Nokia emulator intersperses more platform-specific screens than the other emulators.

Microsoft Mobile Explorer 3.0 (XP Large Screen)

Figures 8.25 through 8.34 are screenshots of a walkthrough of the Mobile Cinema application run using the Microsoft Mobile Explorer 3.0 XP emulator.

Figure 8.25 Home Page

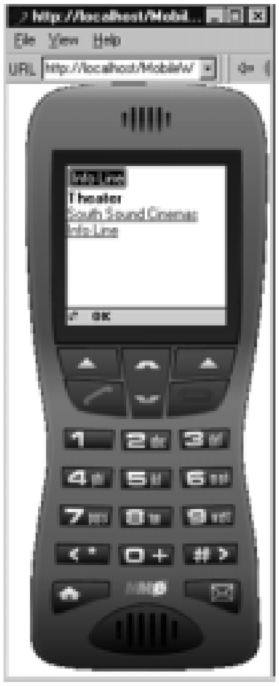


Figure 8.26 Menu

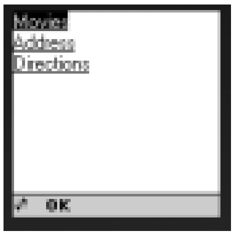

Movies
Address
Directions

Figure 8.27 Address

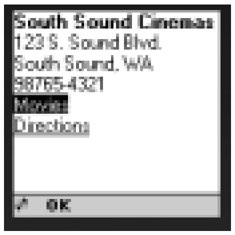
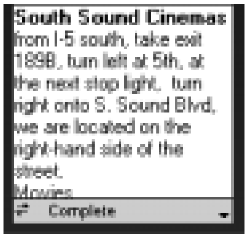
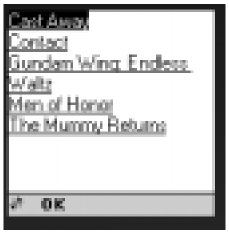

South Sound Cinemas
123 S. Sound Blvd.
South Sound, WA
98765-4321
Movies
Directions

Figure 8.28 Directions



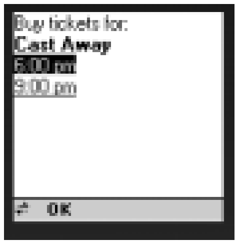
South Sound Cinemas
from I-5 south, take exit 189B, turn left at 5th, at the next stop light, turn right onto S. Sound Blvd, we are located on the right-hand side of the street.
Movies
Complete

Figure 8.29 Movies



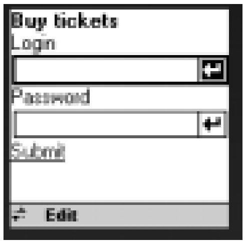
Cast Away
Contact
Gundam Wing: Endless Waltz
Men of Honor
The Mummy Returns
OK

Figure 8.30 Showtimes



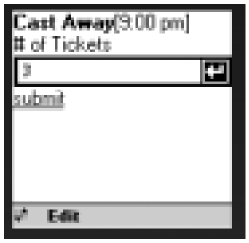
Buy tickets for:
Cast Away
6:00 pm
9:00 pm
OK

Figure 8.31 Login



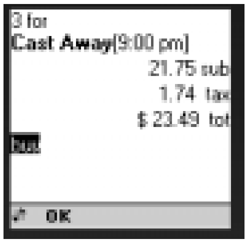
Buy tickets
Login
Password
Submit
Edit

Figure 8.32 Ticket Count



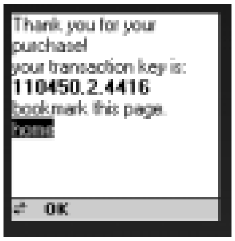
Cast Away[9:00 pm]
# of Tickets
3
submit
Edit

Figure 8.33 Recap



3 for
Cast Away[9:00 pm]
21.75 sub
1.74 tax
$ 23.49 tot
buy
OK

Figure 8.34 Receive *TransKey*



Thank you for your purchase!
your transaction key is:
110450.2.4416
bookmark this page.
home
OK

Microsoft Mobile Explorer 3.0 (Small Screen)

Figures 8.35 through 8.57 are screenshots of a walkthrough of the Mobile Cinema application run using the Microsoft Mobile Explorer 3.0 emulator.

Figure 8.35 Home Page

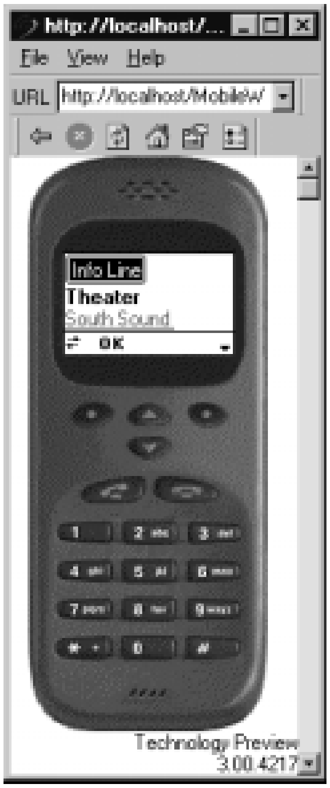


Figure 8.36 Menu

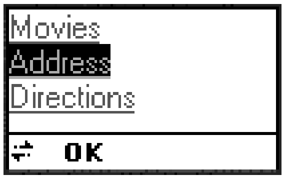


Figure 8.37 Address Top

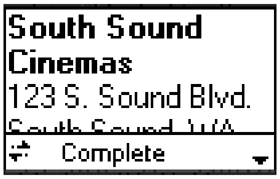
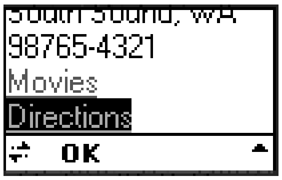
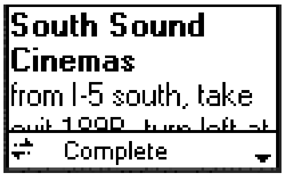

South Sound Cinemas
123 S. Sound Blvd.

Figure 8.38 Address Bottom



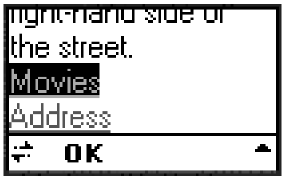
South Sound, WA
98765-4321
Movies
Directions
OK

Figure 8.39 Directions Top



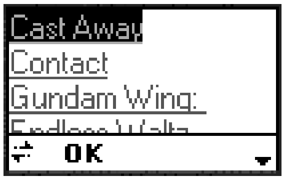
South Sound Cinemas
from I-5 south, take
exit 109B, turn left at
Complete

Figure 8.40 Directions Bottom



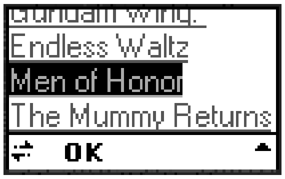
right-hand side of
the street.
Movies
Address
OK

Figure 8.41 Movies Top



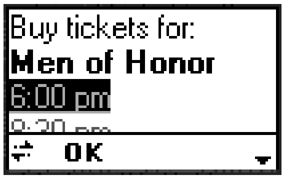
Cast Away
Contact
Gundam Wing:
Endless Waltz
OK

Figure 8.42 Movies Bottom



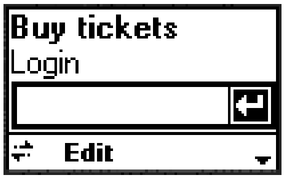
Gundam Wing:
Endless Waltz
Men of Honor
The Mummy Returns
OK

Figure 8.43 Showtimes



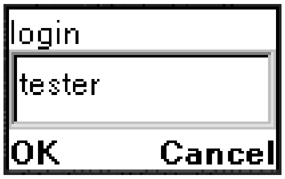
Buy tickets for:
Men of Honor
6:00 pm
8:20 pm
OK

Figure 8.44 Login Top



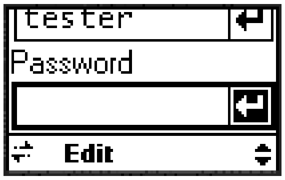
Buy tickets
Login
Edit

Figure 8.45 Login Edit Mode Top



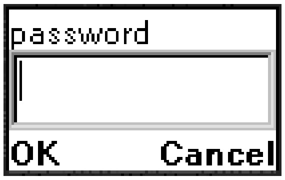
login
tester
OK Cancel

Figure 8.46 Login Bottom



tester
Password
Edit

Figure 8.47 Login Edit Mode Bottom



password
OK Cancel

Figure 8.48 Login Submit

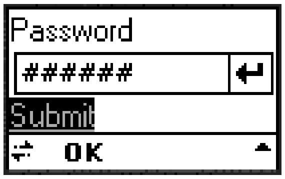

Password
######
Submit
OK

Figure 8.49 Ticket Count Top

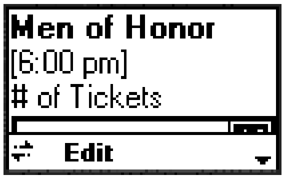

Men of Honor
[6:00 pm]
# of Tickets
Edit

Figure 8.50 Ticket Count Bottom

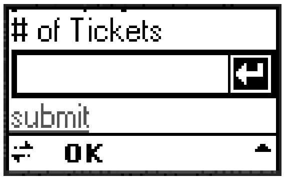

# of Tickets
submit
OK

Figure 8.51 Ticket Edit Mode

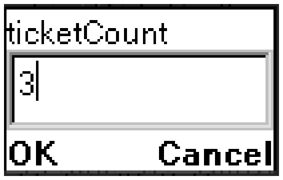

ticketCount
3
OK Cancel

Figure 8.52 Ticket Count Submit

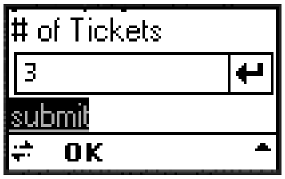

# of Tickets
3
submit
OK

Figure 8.53 Recap Top

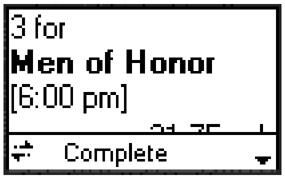

3 for
Men of Honor
[6:00 pm]
Complete

Figure 8.54 Recap Data

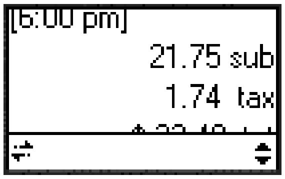

[6:00 pm]
21.75 sub
1.74 tax

Figure 8.55 Recap Bottom

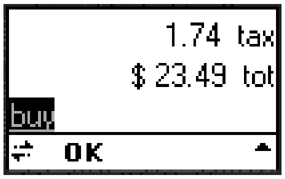

1.74 tax
$ 23.49 tot
buy
OK

Figure 8.56 Receive *TransKey* Top

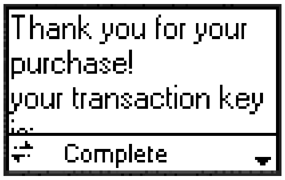
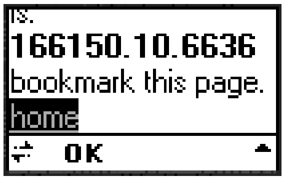

Thank you for your purchase!
your transaction key
Complete

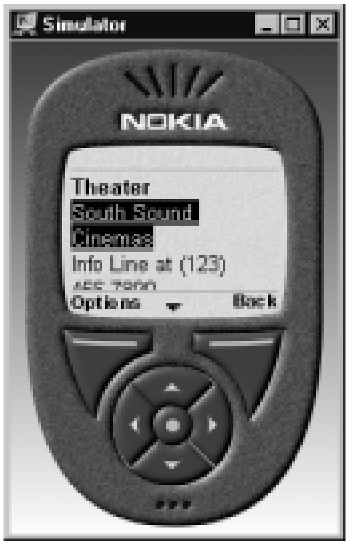
Figure 8.57 Receive *TransKey* Bottom


is:
166150.10.6636
bookmark this page.
home
OK

Nokia 7110 Emulator

Figures 8.58 through 8.81 are screenshots of a walkthrough of the Mobile Cinema application run using the Nokia 7110 WAP emulator.

Figure 8.58 Home Page



Theater
South Sound Cinemas
Info Line at (123)

Figure 8.59 Menu

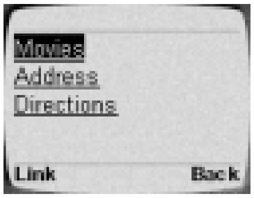
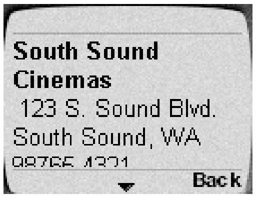
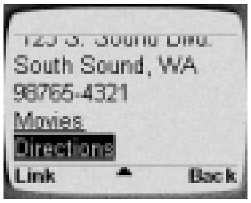


Figure 8.60 Address Top



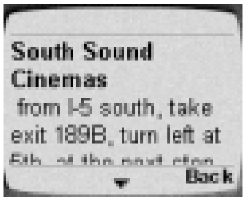
South Sound Cinemas
123 S. Sound Blvd.
South Sound, WA

Figure 8.61 Address Bottom



South Sound, WA
98765-4321

Figure 8.62 Directions Top



South Sound Cinemas
from I-5 south, take exit 189B, turn left at

Figure 8.63 Directions Bottom

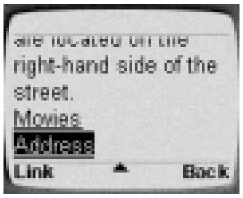

are located on the
right-hand side of the
street.
Movies
Address
Link Back

Figure 8.64 Movies Top

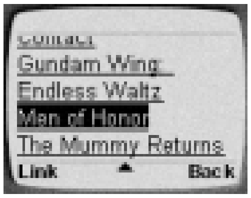

Contact
Gundam Wing:
Endless Waltz
Men of Honor
The Mummy Returns
Link Back

Figure 8.65 Showtimes

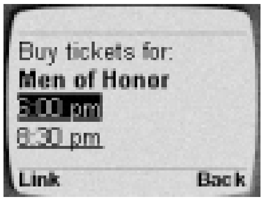

Buy tickets for:
Men of Honor
6:00 pm
8:30 pm
Link Back

Figure 8.66 Login Page

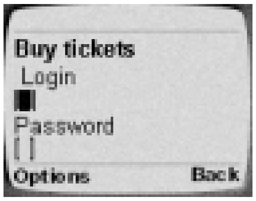

Buy tickets
Login
[ ]
Password
[ ]
Options Back

Figure 8.67 Login Edit Mode

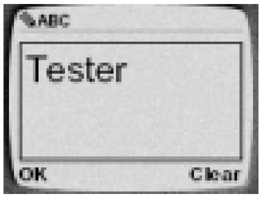

ABC
Tester
OK Clear

Figure 8.68 Login Page with First Field

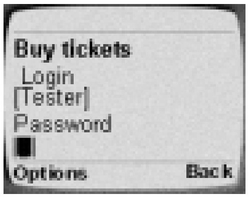

Buy tickets
Login
[Tester]
Password
[ ]
Options Back

Figure 8.69 Password Edit Mode

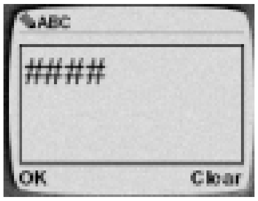

ABC
#####
OK Clear

Figure 8.70 Login with User Data

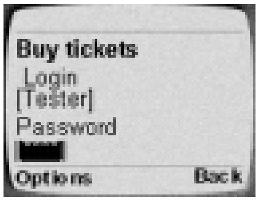

Buy tickets
Login
[Tester]
Password
Options Back

Figure 8.71 Login Option Screen

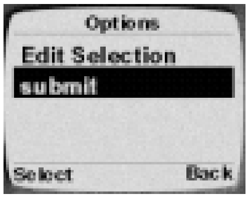

Options
Edit Selection
submit
Select Back

Figure 8.72 Login Submit Redirect Screen

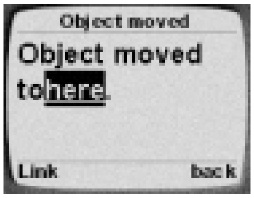
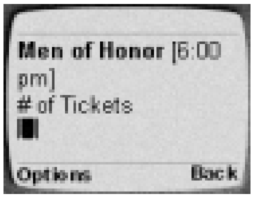
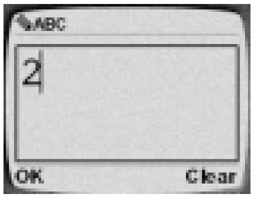

Object moved
Object moved
to here.
Link back

Figure 8.73 Ticket Count



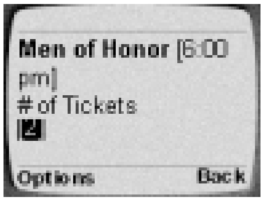
Men of Honor [6:00 pm]
# of Tickets

Figure 8.74 Ticket Count Edit Mode



2

Figure 8.75 Ticket Count User Data



Men of Honor [6:00 pm]
# of Tickets
2

Figure 8.76 Ticket Count Option Screen

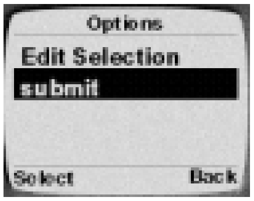
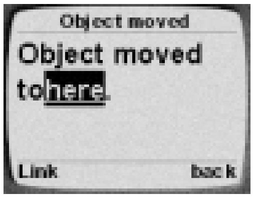
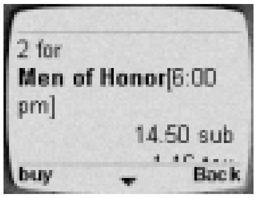


Figure 8.77 Ticket Count Submit Redirect Screen



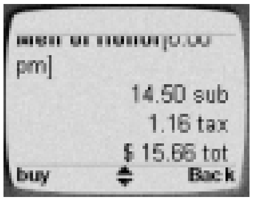
Object moved
Object moved to here.

Figure 8.78 Recap Top



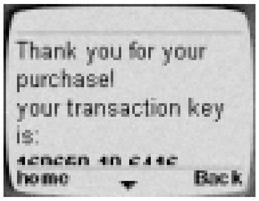
2 for
Men of Honor[6:00 pm]
14.50 sub

Figure 8.79 Recap Bottom



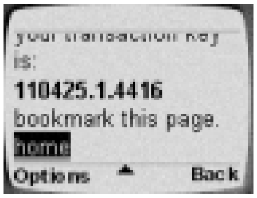
pm]
14.50 sub
1.16 tax
$ 15.66 tot

Figure 8.80 Receive *TransKey* Top



Thank you for your purchase!
your transaction key is:

Figure 8.81 Receive *TransKey* Bottom

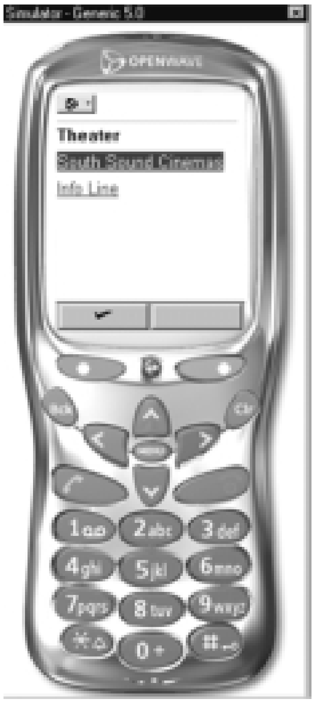


is:
110425.1.4416
bookmark this page.
home

Openwave 5.0 WAP Emulator

Figures 8.82 through 8.92 are screenshots of a walkthrough of the Mobile Cinema application run using the Openwave 5 WAP emulator.

Figure 8.82 Home Page (Images courtesy of Openwave Systems, Inc.)



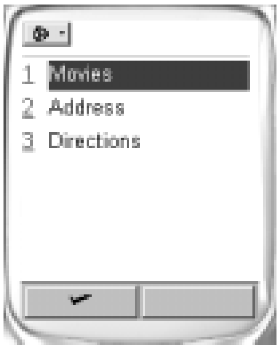
Simulator - Generic 5.0

Theater

South Sound Cinemas

Info Line

Figure 8.83 Menu

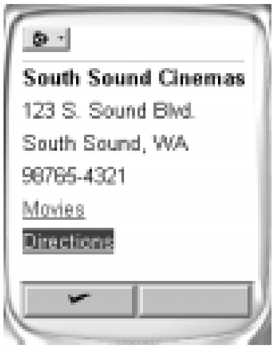


1 Movies

2 Address

3 Directions

Figure 8.84 Address



South Sound Cinemas

123 S. Sound Blvd.

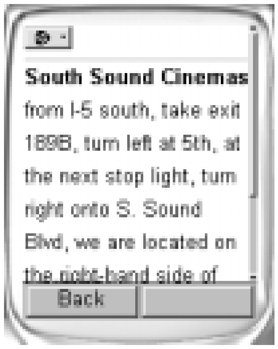
South Sound, WA

98765-4321

Movies

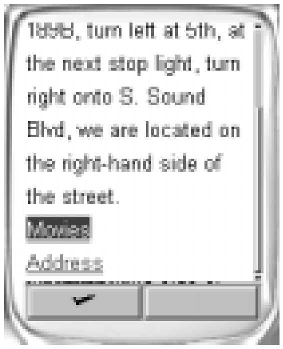
Directions

Figure 8.85 Directions Top



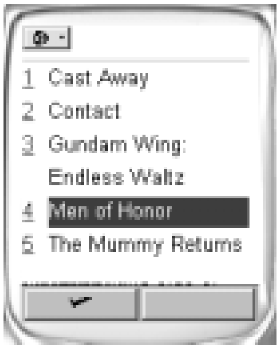
South Sound Cinemas
from I-5 south, take exit 189B, turn left at 5th, at the next stop light, turn right onto S. Sound Blvd, we are located on the right-hand side of

Figure 8.86 Directions Bottom



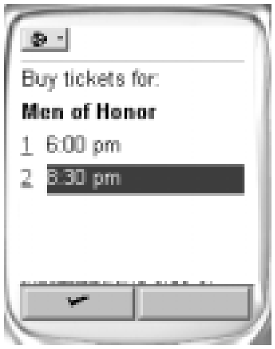
189B, turn left at 5th, at the next stop light, turn right onto S. Sound Blvd, we are located on the right-hand side of the street.
Movies
Address

Figure 8.87 Movies



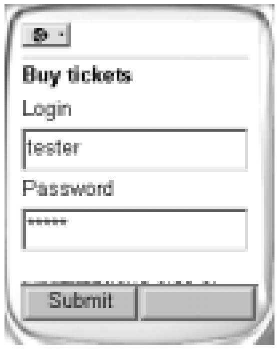
1 Cast Away
2 Contact
3 Gundam Wing: Endless Waltz
4 Men of Honor
5 The Mummy Returns

Figure 8.88 Showtimes



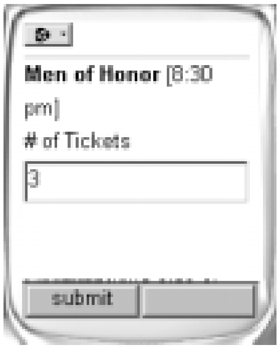
Buy tickets for:
Men of Honor
1 6:00 pm
2 8:30 pm

Figure 8.89 Login



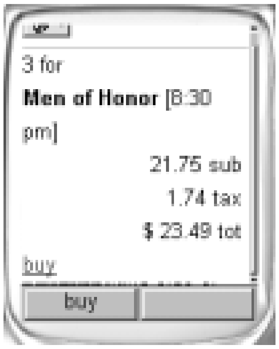
Buy tickets
Login
tester
Password
******

Figure 8.90 Ticket Count



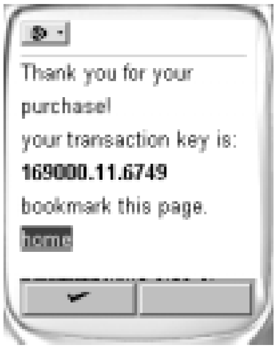
Men of Honor [8:30 pm]
# of Tickets
3

Figure 8.91 Recap



3 for
Men of Honor [8:30 pm]
21.75 sub
1.74 tax
$ 23.49 tot
buy

Figure 8.92 Receive *TransKey*



Thank you for your purchase!
your transaction key is:
169000.11.6749
bookmark this page.
home

Siemens S45 WAP Emulator

Figures 8.93 through 8.108 are screenshots of a walkthrough of the Mobile Cinema application run using the Siemens S45 WAP emulator.

Figure 8.93 Home Page

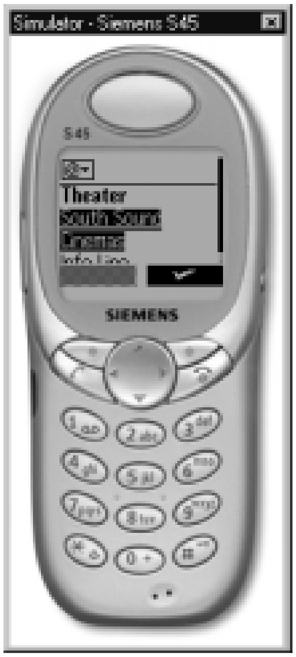
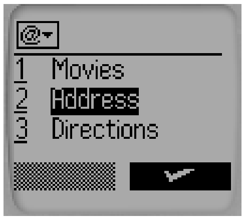
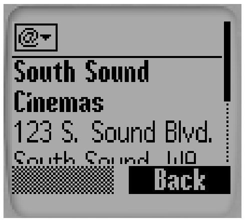


Figure 8.94 Menu



1 Movies
2 Address
3 Directions

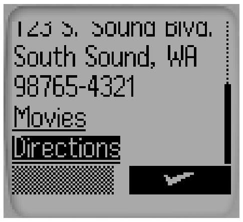
Figure 8.95 Address



South Sound Cinemas
123 S. Sound Blvd.
South Sound, WA

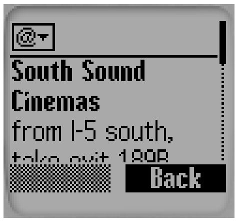
Back

Figure 8.96 Address Bottom



123 S. Sound Blvd.
South Sound, WA
98765-4321
Movies
Directions

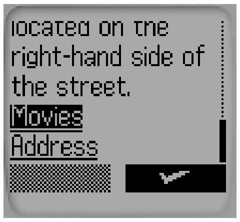
Figure 8.97 Directions



South Sound Cinemas
from I-5 south,
take exit 180B

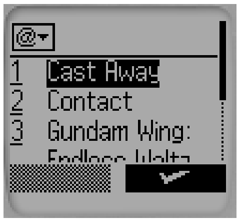
Back

Figure 8.98 Directions Bottom



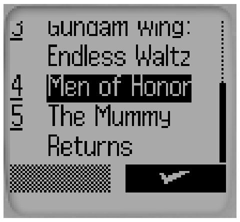
located on the
right-hand side of
the street.
Movies
Address

Figure 8.99 Movies



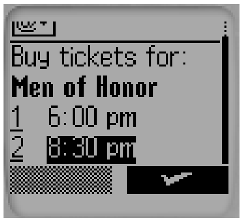
1 Cast Away
2 Contact
3 Gundam Wing:
Endless Waltz

Figure 8.100 Movies Bottom



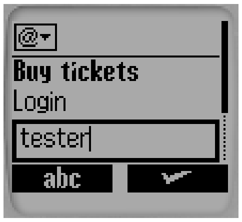
3 Gundam Wing:
Endless Waltz
4 Men of Honor
5 The Mummy
Returns

Figure 8.101 Showtimes



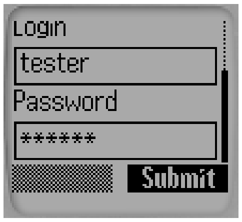
Buy tickets for:
Men of Honor
1 6:00 pm
2 8:30 pm

Figure 8.102 Login Page



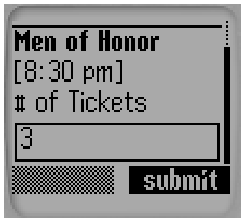
Buy tickets
Login
tester
abc

Figure 8.103 Login Submit



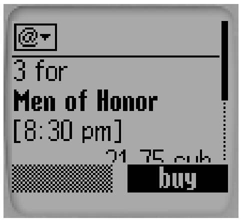
Login
tester
Password
******
Submit

Figure 8.104 Ticket Count



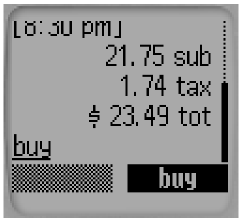
Men of Honor
[8:30 pm]
# of Tickets
3
submit

Figure 8.105 Recap



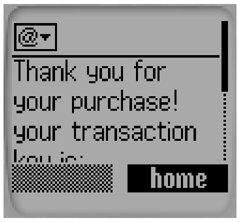
3 for
Men of Honor
[8:30 pm]
21.75 sub
buy

Figure 8.106 Recap Bottom



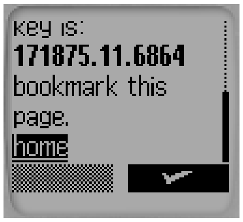
[8:30 pm]
21.75 sub
1.74 tax
$ 23.49 tot
buy
buy

Figure 8.107 Receive *TransKey*



Thank you for
your purchase!
your transaction
key is:
home

Figure 8.108 Receive *TransKey* Bottom



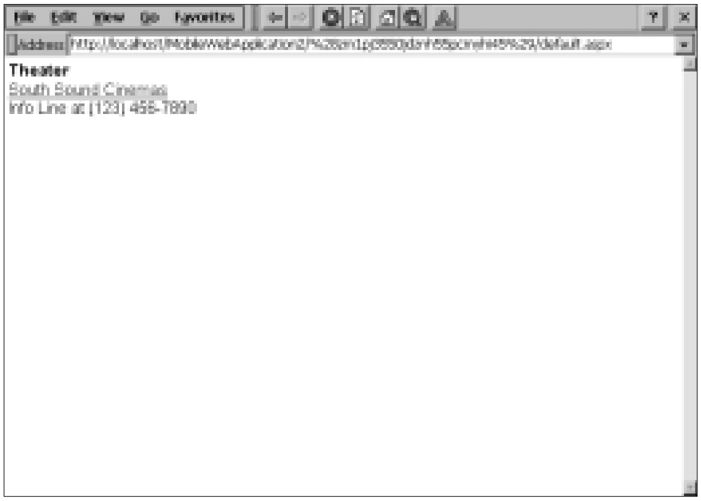
key is:
171875.11.6864
bookmark this
page.
home

Windows CE Platform/Handheld PC Emulator (Internet Explorer)

Figures 8.109 through 8.118 are screenshots of a walkthrough of the Mobile Cinema application run using the Windows CE Platform SDK/Desktop

Handheld PC Pro Emulator running Internet Explorer (IE). Although the UI for these pages is rather bland for a standard IE page, the interesting point is that the mobile Web forms do render standard HTML for IE/WinCE as well as for standard Web browsers. Also note that user controls can include device-specific code that will enable the developer to create a more robust UI for specific devices.

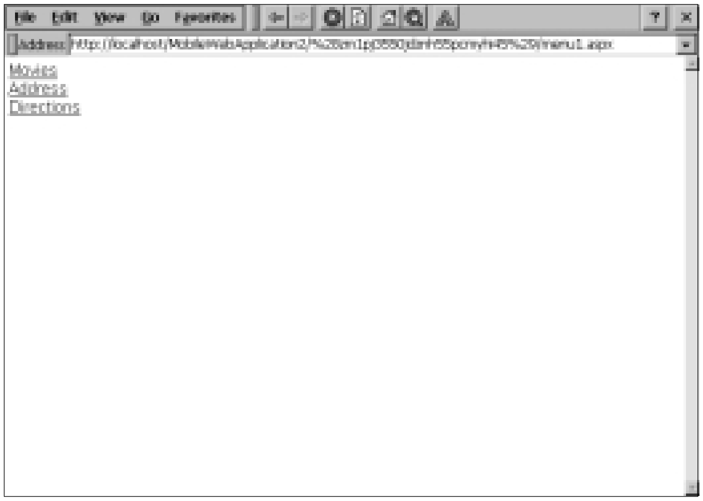
Figure 8.109 Home Page



http://localhost/MobileWebApplication2/(%28am1pj3550dznh55pcnehi45%29)/default.aspx

Theater
South Sound Cinemas
Info Line at (123) 456-7890

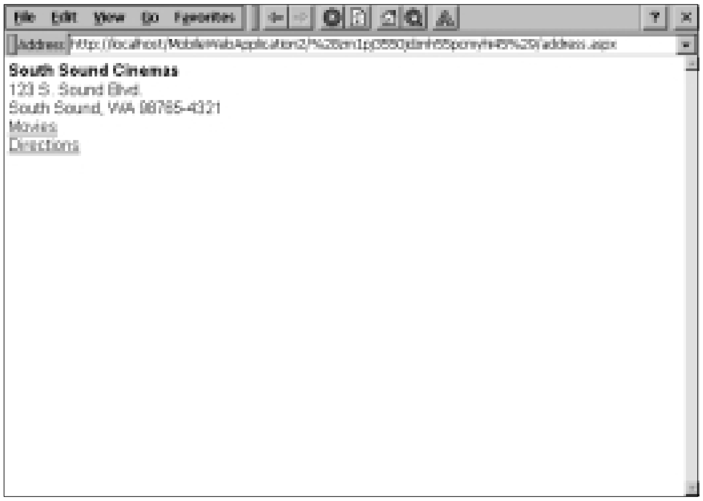
Figure 8.110 Menu



http://localhost/MobileWebApplication2/(%28am1pj3550dznh55pcnehi45%29)/menu1.aspx

Movies
Address
Directions

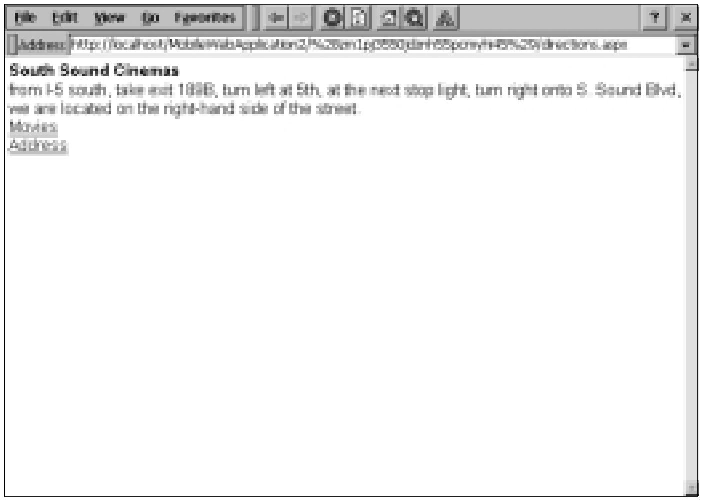
Figure 8.111 Address



http://localhost/MobileWebApplication2/%28zm1p0550jdnh55pcny4n45%29/address.aspx

South Sound Cinemas
123 S. Sound Blvd.
South Sound, WA 98765-4321
Movies
Directions

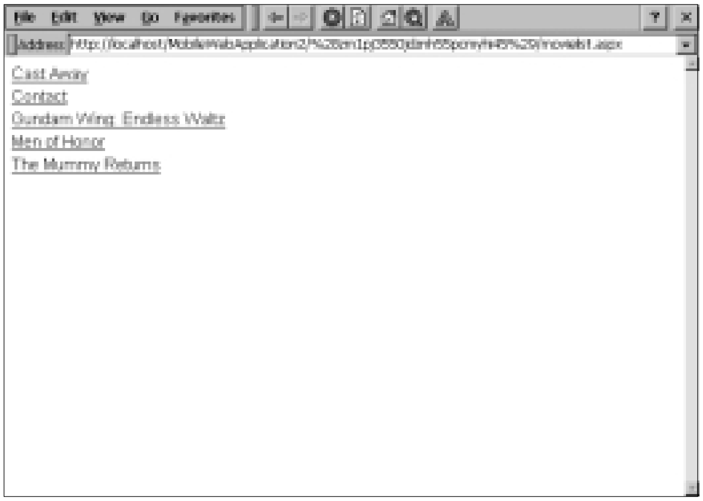
Figure 8.112 Directions



http://localhost/MobileWebApplication2/%28zm1p0550jdnh55pcny4n45%29/directions.aspx

South Sound Cinemas
from I-5 south, take exit 189B, turn left at 5th, at the next stop light, turn right onto S. Sound Blvd, we are located on the right-hand side of the street.
Movies
Address

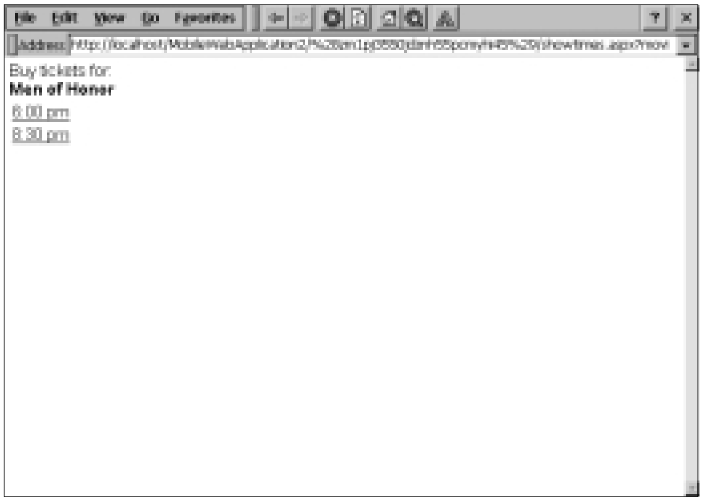
Figure 8.113 Movies



http://localhost/MobileWebApplication2/%28om1pj3550jdanh55pcmyh45%29/movielist.aspx

Cast Away
Contact
Gundam Wing: Endless Waltz
Men of Honor
The Mummy Returns

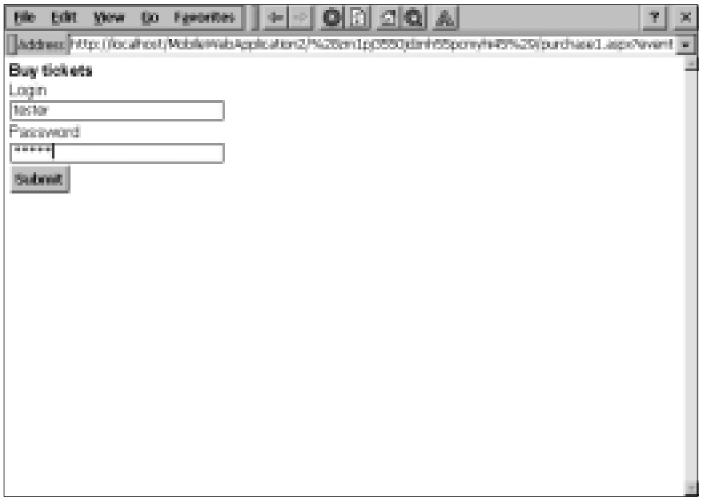
Figure 8.114 Showtimes



http://localhost/MobileWebApplication2/%28om1pj3550jdanh55pcmyh45%29/showtimes.aspx?mov

Buy tickets for:
Men of Honor
6:00 pm
8:30 pm

Figure 8.115 Login



http://localhost/MobileWebApplication2/(%28zm1p[illegible]%29)/purchase1.aspx?event

Buy tickets

Login

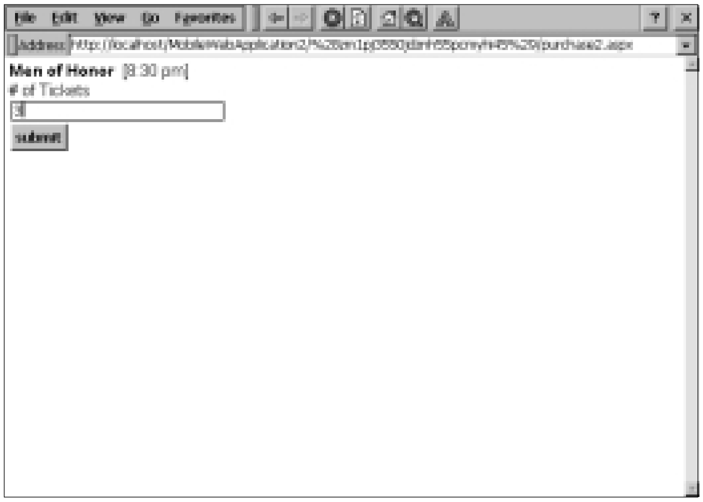
tester

Password

******

Submit

Figure 8.116 Ticket Count



http://localhost/MobileWebApplication2/(%28zm1p[illegible]%29)/purchase2.aspx

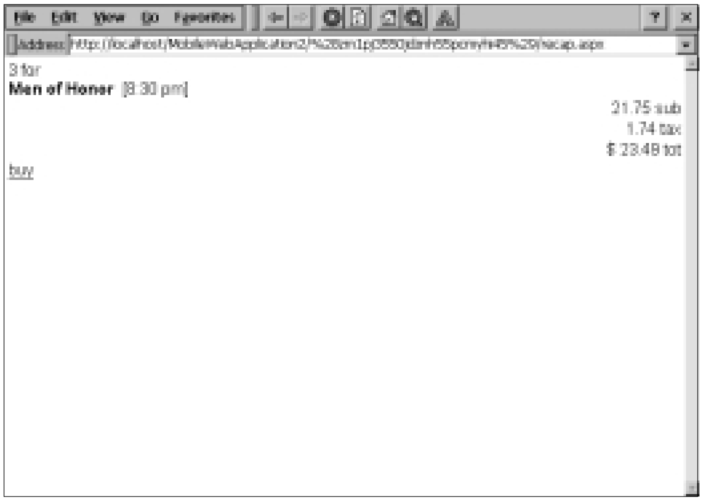
Men of Honor [8:30 pm]

# of Tickets

3

submit

Figure 8.117 Recap

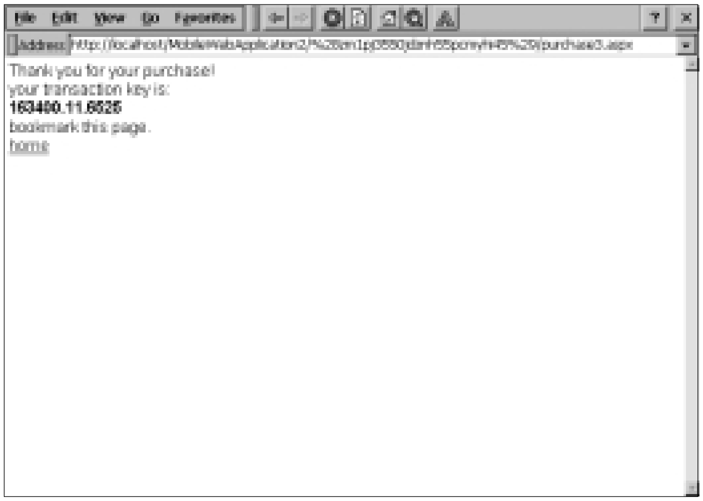

File
Edit
View
Go
Favorites
Address
http://localhost/MobileWebApplication2/%28xm1pj3550jdanh55pcnyhi45%29/recap.aspx

3 for
Men of Honor [8:30 pm]

21.75 sub
1.74 tax
$ 23.49 tot

buy

Figure 8.118 Receive *TransKey*


File
Edit
View
Go
Favorites
Address
http://localhost/MobileWebApplication2/%28xm1pj3550jdanh55pcnyhi45%29/purchase3.aspx

Thank you for your purchase!
your transaction key is:
163400.11.6526
bookmark this page.
home

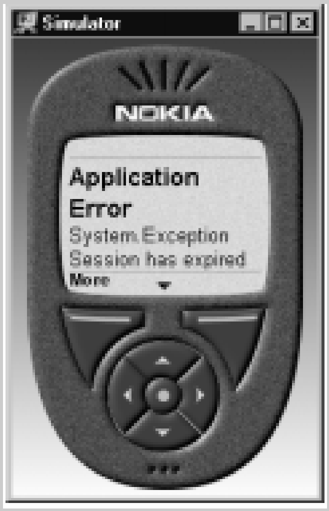
Debugging…

Some Mobile Devices Do Not Support Cookies

If one or more of the mobile devices you need to support do not support cookies, your ASP.NET pages may not function correctly. You may get errors stating that your session has expired or the client did not send a valid cookie method (see Figures 8.119 and 8.120). You can easily fix this problem:

1. Open Web.config.
2. Find the element *sessionState*.
3. Set the *sessionState* attribute *cookieless = "true"*.
4. Save and rebuild your project.

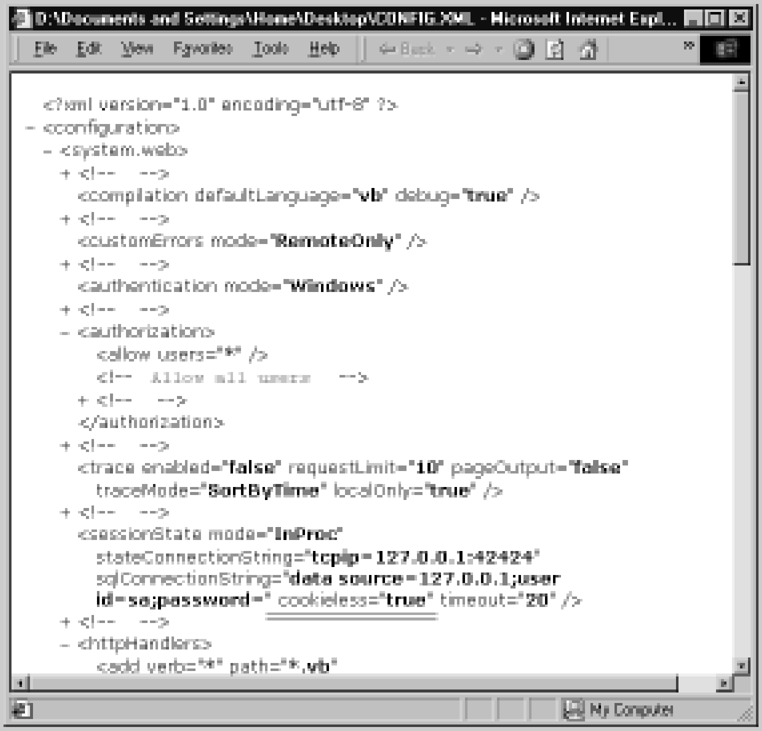
Figure 8.119 Session Errors Generated by Devices That Do Not Support Cookies



Application Error
System.Exception
Session has expired
More

Continued

Figure 8.120 The Web.config File


D:\Documents and Settings\Home\Desktop\CONFIG.XML - Microsoft Internet Expl...
File Edit View Favorites Tools Help
Back
My Computer

```
  <?xml version="1.0" encoding="utf-8" ?>
- <configuration>
  - <system.web>
    + <!--   -->
      <compilation defaultLanguage="vb" debug="true" />
    + <!--   -->
      <customErrors mode="RemoteOnly" />
    + <!--   -->
      <authentication mode="Windows" />
    + <!--   -->
    - <authorization>
        <allow users="*" />
        <!--  Allow all users   -->
      + <!--   -->
      </authorization>
    + <!--   -->
      <trace enabled="false" requestLimit="10" pageOutput="false"
        traceMode="SortByTime" localOnly="true" />
    + <!--   -->
      <sessionState mode="InProc"
        stateConnectionString="tcpip=127.0.0.1:42424"
        sqlConnectionString="data source=127.0.0.1;user
        id=sa;password=" cookieless="true" timeout="20" />
    + <!--   -->
    - <httpHandlers>
        <add verb="*" path="*.vb"
```

Deployment

Deployment for your Mobile application is as simple as deploying a basic Web application. You can use the built-in service that comes with VS.NET. Simply copy Web from the solutions explorer window and input the information to the remote server you are to upload to. You can transfer files with FrontPage server extensions, or you can FTP directly. The Web directory must have a bin folder and the DLL must be inside.

Summary

In this chapter, we stated the target devices your application would support and gave you links to obtain their emulators. We explained code design tradeoffs that were necessary to realize this goal: single form pages, avoidance of the *ObjectList* control, and configuring the site to run without cookies. You developed a scalable process that employs a relatively simple schema and a data access component for all database interactions, with built-in transaction handling through SQL stored procedures. You also stubbed out and simulated interfaces to third-party services, such as online wallets and passport style authentication, while providing insight into the effective use of mobile Web forms to render the UI on various device emulators. We focused on the following Mobile Web Form controls:

- *Mobile:Label*
- *Mobile:TextView*
- *Mobile:List*
- *Mobile:Call*

We also demonstrated the following:

- Setting control properties directly in their element attributes as well as through using the object model in code-behind pages.
- Accessing data from static XML files to accessing data from SQL stored procedures.
- Binding the *List* control to a *DataSet* and setting and updating the properties of the generated code.
- How different devices handle the same display information.

Although this case study is rather simple, you should note that developing this type of system as a real-life application would require substantial investment on the customer's part, in financial and process reorganization, as well as legwork in setting up online wallet, authentication, and bank/merchant transaction processing. Hosting the site from a central location that has access to all local theaters' data might also make sense.

Solutions Fast Track

System Design/Flow

☑ Identify and diagram the scenario you need to support, using flow charts to assist in the conceptual design of the database schema as well as candidate stored procedures.

☑ Storyboard the process so that all steps and screens can be laid out and evaluated before building the solution.

The Data

☑ Creating stored procedures is the best solution for performance as opposed to ad hoc queries.

☑ Using the roll back trans command in SQL ensures that the transaction will not be committed unless steps within the transaction have completed successfully.

☑ In SQL, you can declare local variables and create program logic to get to a solution within a stored procedure.

Designing the Interface

☑ Know what target platforms or devices you must support, be sure to test each screen developed in emulators for each. Understand the impact of tradeoffs made for device compatibility. In the application developed for this chapter, multiple WML cards per page were traded for Pocket IE compatibility. Also the *ObjectList* control was avoided for this same reason.

☑ Limit the size of the UI; that is, keep it small and concise.

☑ For more control over individual device rendering of content, look into device filters and using the Mobile Capabilities component (a component similar to the ASP.NET *HttpBrowserCapabilities* component and the earlier *MSWC.Browser* component used in standard ASP).

☑ For scalability and code reuse, create stored procedures and data access components to handle the interaction between the page and the database.

☑ Use the *TextView* control instead of the *Label* control when you need to pass markup with text.

☑ Remember, not all mobile devices can handle cookies; if you think you may have to use any session variables or are not sure if some of your code is relying on cookies, switch the Web.config file's *sessionState* attribute *cookieless* to true.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: Does the mobile framework have form validation controls similar to ASP.NET Web forms?

A: Yes. However, this is beta software and these controls still need some work.

Q: Do I need to learn WML to develop with Mobile .NET?

A: No, almost all of the code is hidden behind an object model. The advantage of this is that as new devices hit the market, you can add new style templates with minimal recoding of applications.

Q: Can I see the WML that is generated by the mobile Web forms?

A: Yes, if you are testing with the Nokia 3.0 Toolkit. It will show you the WML that the phone emulator is rendering. However, this is not currently a feature in VS.NET Beta 2.

Index