Programming .NET Web Services

Alex Ferrara & Matthew MacDonald
Web services are a promising new technology for delivering a variety of consumer and business services over the Internet, from the Google search engine, Microsoft Map Point, and eBay Price Watcher, to B2B back-end systems like Microsoft Passport and in-house integration solutions. Developers are finding Microsoft's .NET Framework and Visual Studio .NET to be powerful tools for creating and deploying web services.

*Programming .NET Web Services* provides experienced developers with a fast-paced and comprehensive tutorial that shows you how to develop web services using ASP.NET, Visual Studio .NET, and the attributes and types of the `System.Web.Services` namespaces of the .NET Framework. Beginning with a close look at the defining web services protocols, especially HTTP, SOAP, and WSDL, *Programming .NET Web Services* quickly introduces the unique features of Visual Studio .NET and the .NET Framework that make creating and deploying web services easy. Filled with numerous code examples in C#, the book leads you through some of the more challenging issues of web services development, including the use of proxies, the marshalling of simple and complex data types, state management, security, performance tuning, and cross-platform interoperability. The book also covers:

- Understanding the myriad files generated by Visual Studio .NET
- Reading key XML schema documents, including SOAP requests and responses, representations of simple and complex data types, and WSDL and UDDI documents
- Using .NET and XML attributes for greater control over web service behavior
- Deploying web services using IIS and FrontPage Extensions or UNC file shares
- Using asynchronous messaging and caching for better performance
- Making your web services accessible to third parties (or within your company) with UDDI publishing protocols and tModels
- Auto-generating and working with client proxy classes using WSDL, VS.NET, and the .NET Framework SDK tools
- Writing secure web services using the security features of the .NET Framework and IIS

The book concludes with an extensive list of additional resources and an API Quick Reference to the `System.Web.Services` namespaces of the .NET Framework Class Library. Written for programmers familiar with C# and the .NET Framework who want to build industrial-strength XML web services, *Programming .NET Web Services* is full of practical information and good advice.
# Table of Contents

## Preface

1. Understanding Web Services ................................................. 1
   - Why Web Services? .................................................. 1
   - The Origin of Web Services ....................................... 4
   - The .NET Web Service Architecture ............................. 10
   - Competing Web Service Technologies ............................ 16
   - Other Technologies ............................................... 17

2. Creating ASP.NET Web Services ......................................... 20
   - Creating a Web Service: “Hello, World” ....................... 20
   - The WebService Attribute ....................................... 37
   - The WebMethod Attribute ....................................... 39
   - Deploying a Web Service ........................................ 45
   - Using the .NET Compilers ....................................... 50

3. Consuming Web Services .................................................. 54
   - The Web Service Consumer Model ............................... 54
   - Accessing Web Services: HTTP .................................. 57
   - Exposing Web Services: WSDL .................................. 68
   - Calling Web Services: the Client Proxy ....................... 78
   - Creating a Client Proxy ........................................ 79
   - Working with the Proxy Class .................................. 87
   - Consuming a Web Site Using a Screen Scraper Service .... 93

4. Working with Data Types .................................................. 101
   - Working with Simple Data Types .............................. 101
   - Working with Complex Data Types ......................... 110
Using Complex Data Types as Arguments  119
Passing Data with DataSets  125
Using Binary Data  132

5. **Managing State** ................................................................. 135
   The Great State Debate  135
   State Management Scenarios  142
   ASP.NET’s Session State Facility  145
   Session State Configuration  148
   Example: Managing Session State  152
   Application State  157
   Custom Cookies  158
   Stateless Web Services and Tickets  160
   SOAP Headers  161

6. **Asynchronous Services** ....................................................... 171
   Asynchronous Consumption  171
   Asynchronous Services  185

7. **Caching and Profiling** .......................................................... 197
   ASP.NET Caching  197
   Simple Output Caching  200
   Data Caching  205
   Cache Dependencies  213
   Replacing Stateful Design with Caching  219
   Profiling Your Web Service  224

8. **Debugging, Tracing, and Logging** ............................................ 230
   Debugging in Visual Studio .NET  230
   Using Exceptions  233
   Logging Errors with .NET  239
   SOAP Extensions  247
   Advanced SOAP Extensions  254

9. **Security and Authentication** .................................................... 262
   Potential Vulnerabilities  262
   Web Service Authentication  264
   IIS Security Features  265
   ASP.NET Security  270
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmatic Security</td>
<td>280</td>
<td></td>
</tr>
<tr>
<td>Custom Authentication Techniques</td>
<td>289</td>
<td></td>
</tr>
<tr>
<td>Process Security</td>
<td>297</td>
<td></td>
</tr>
<tr>
<td>10. Publishing and Discovery</td>
<td>299</td>
<td></td>
</tr>
<tr>
<td>Publishing</td>
<td>299</td>
<td></td>
</tr>
<tr>
<td>UDDI</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>UDDI Business Registries</td>
<td>302</td>
<td></td>
</tr>
<tr>
<td>Registering a Business Entity</td>
<td>305</td>
<td></td>
</tr>
<tr>
<td>WSDL and UDDI</td>
<td>309</td>
<td></td>
</tr>
<tr>
<td>Visual Studio .NET and UDDI</td>
<td>321</td>
<td></td>
</tr>
<tr>
<td>UDDI Messaging</td>
<td>322</td>
<td></td>
</tr>
<tr>
<td>UDDI Access with .NET</td>
<td>326</td>
<td></td>
</tr>
<tr>
<td>The WS-Inspection Alternative</td>
<td>330</td>
<td></td>
</tr>
<tr>
<td>Customizing the IE Test Page</td>
<td>332</td>
<td></td>
</tr>
<tr>
<td>11. Interoperability</td>
<td>334</td>
<td></td>
</tr>
<tr>
<td>State of the Specifications</td>
<td>334</td>
<td></td>
</tr>
<tr>
<td>Implementation Differences</td>
<td>336</td>
<td></td>
</tr>
<tr>
<td>SOAP Interoperability</td>
<td>340</td>
<td></td>
</tr>
<tr>
<td>WSDL Interoperability</td>
<td>342</td>
<td></td>
</tr>
<tr>
<td>HTTP Interoperability and M-Post</td>
<td>343</td>
<td></td>
</tr>
<tr>
<td>Known Interoperability Problems</td>
<td>343</td>
<td></td>
</tr>
<tr>
<td>Maximizing Interoperability</td>
<td>346</td>
<td></td>
</tr>
<tr>
<td>A. Namespace Quick Reference</td>
<td>349</td>
<td></td>
</tr>
<tr>
<td>B. Web Service Technologies</td>
<td>376</td>
<td></td>
</tr>
<tr>
<td>Index</td>
<td>383</td>
<td></td>
</tr>
</tbody>
</table>
Over the last few years, we’ve lived with an endless stream of web-related hype. Technology evangelists, columnists, and CEOs repeatedly proclaimed that the Internet was about to revolutionize everything from the social fabric of our entire society to grocery shopping. Some of these changes have taken place, but other radical predictions and ambitious new technologies now look—well, a little embarrassing.

One of the visions that has been conjured up time and time again that of a “programmable Internet,” on which all sorts of applications and devices communicate continuously, sharing features and functionality openly and painlessly. The truth is, though we now have an unprecedented network that links computers across the globe, we haven’t begun to realize its full potential. Despite ambitious technologies like COM/DCOM, CORBA/IIOP, RMI, and XML-RPC, the Internet is currently used primarily for sending email and retrieving HTML pages; programs on proprietary platforms remain unable to use each others services without costly customization.

Now Microsoft (and traditional rivals like IBM and Sun) have proposed a new “web services platform,” based on XML and related technologies that aims to let applications share functionality over the Internet as easily as they can on a local computer. The potential is amazing, and the implementation is simple, elegant, and surprisingly open. XML is incorporated from the ground up, cross-platform support is inherent, and productivity tools are available from participating vendors to handle all the heavy lifting. Web services and simple web services directories are already spreading over the Internet, with everything from horoscope and translation services to Microsoft’s own TerraService, which provides access to a 1.5-terabyte SQL Server database of satellite images. Companies like eBay and McAfee have been hard at work developing innovative web service solutions with .NET, and key sites such as Google and Amazon have already exposed some of their capabilities using web services interfaces.

The Microsoft .NET Framework is both an implementation of this platform and a means to easily build services that can interoperate on it. Microsoft has (at least to some degree) abandoned its traditional preference for closed technologies and proprietary standards. Java clients on Unix machines can consume .NET services, and .NET
clients can interact with web services written in most other non-.NET languages just as readily, provided they adhere to emerging XML-based web services standard protocols like SOAP and WSDL. The complexity and headaches that have clouded distributed programming with COM and CORBA have been replaced with simple, lightweight XML-based standards. To sum it all up: XML-based web services in general, and .NET web services in particular, are a radically changed and improved attempt to reach a long-standing goal of universal connectivity.

What This Book Covers

This book explores Microsoft .NET web services from the developer’s perspective. We explain how to create web services with the .NET Framework and its ASP.NET tools, but more importantly, we consider the best practices needed to design web services that are efficient, scalable, and robust. This book also sheds some light on the underlying plumbing for the .NET web service technology: the open standards like SOAP and WSDL that are at work behind the scenes. This understanding is not required to write a web service with .NET, but it can be quite useful when deciding how to use its features and how to deal with its limitations.

Throughout this book we’ll also show you how to extend web services. One of the greatest features of web services protocols is their simplicity. Unlike earlier distributed object technologies, XML-based web services standards do not yet specify mechanisms for handling security, transactions, or object pooling. In order to provide these features, you need to design and implement your own solutions and leverage platform technologies like COM+ and ASP.NET. Throughout this book, you’ll see the best ways to develop these homegrown solutions.

Audience

Just as important as what this book covers is what this book doesn’t cover—the syntax of the C# programming language, the fundamentals of ASP.NET, or the basics of the .NET Framework. This book is targeted at the professional developer who already has some .NET experience. If you aren’t already familiar with C# and .NET, there are plenty of other excellent .NET books available from O’Reilly to get you started, including titles like Programming C#, Second Edition, by Jesse Liberty (2002), Programming ASP.NET, by Liberty and Hurwitz (2002) .NET Framework Essentials, Second Edition, by Thai and Lam (2002), and C# in a Nutshell, by Drayton, Albahari, and Neward (2002).

On the other hand, if you are a seasoned .NET developer, you don’t need to worry about wasting time revisiting the basics about types, metadata, and the Common Language Runtime. Instead, our discussion will focus exclusively on web service topics. For example, you’ll see how services work with disconnected DataSets, but we won’t review the basics of ADO.NET. This concentrated focus makes this book
unique among the current crop of .NET books, most of which try to teach the .NET Framework fundamentals, the C# or VB .NET programming language, and other programming basics in one muddled combination.

If you are familiar with the .NET platform, you may have already realized that C# code can be converted to equivalent VB.NET statements on a line-by-line basis. Unfortunately, due to space limitations, the examples in this book are presented in C# code only. Any potential differences that could affect the VB.NET programmer are explained in the text, and VB.NET developers will still find this work to be an excellent reference for web service fundamentals and best practices for web service design. For additional information, see C# and VB.NET Conversion Pocket Reference, by Jose Mojica (O’Reilly, 2002).

What You Need to Use This Book

To make the best use of this book, you’ll need the following software:

• Windows 2000 Professional or Windows XP (the minimum requirement for developing with the .NET Framework).

• The Internet Information Services (IIS) component of Windows. This feature is part of the standard Windows operating system, but it is not installed by default. Ideally, you should install it before you install the .NET Framework. To ensure that it is present, choose Windows Components from the Add or Remove Programs window.

• The .NET Framework, either in its redistributable (SDK) form or as a part of the Visual Studio .NET retail package. You can also work with the second beta of the .NET Framework, although the released version is recommended. The first beta, however, is not supported (or compatible with the examples presented here).

The web service features of Visual Studio .NET (VS.NET) are documented throughout the book, and you may well find that its automatic error-checking, integrated debugging, and IntelliSense statement completion are impossible to live without. However, .NET web services can be coded just as easily by hand and compiled with the command-line csc.exe compiler. This book describes this approach as well. For the most part, we focus on pure code and the technology behind web services. The development approach you use will be of secondary importance. If you do use VS. NET, we assume that you have it installed and set up correctly.

How This Book Is Organized

The book begins by describing web service basics: how they work, how to create them, and how to consume them in a client. Thanks to .NET development tools,
writing a simple web service is not much more difficult than creating an ordinary class, once you know the rules and restrictions. The bulk of the book—the next six chapters—covers ways you can integrate commonly required advanced features into web services. Often, technology details are introduced to solve a particular problem, not up front when they may just cloud a discussion with additional details. For example, SOAP headers make an appearance as a powerful way to implement custom state management, and SOAP extensions appear as a way to perform automatic tracing.

The book ends by considering how web services can be published so they are easily accessible and how you can break down the boundaries between your web services and non-.NET code for truly cross-platform solutions.

This brief outline explains what each chapter contains:

Chapter 1, *Understanding Web Services*

History is littered with failed web technologies, and Microsoft’s contribution to the pile is far from modest, including such legacy approaches as ActiveX documents and Visual Basic web classes. With .NET, however, there’s good reason to believe that Microsoft has learned from the disasters of the past. In this chapter, you’ll learn why .NET XML web services should work where other technologies have stalled and what roles web services will play in the applications of the future. You’ll also learn about the standards, like WSDL, SOAP, and HTTP, that underlie .NET web services.

Chapter 2, *Creating ASP.NET Web Services*

This chapter starts with every developer’s favorite example, the canonical Hello-World program—this time recast as a web service. Before the chapter is finished, you’ll move up to a DNS lookup service and learn how to test your web methods using HTTP GET in Internet Explorer. Along the way, you’ll learn about .NET’s web service types, virtual directories and web service deployment, and how to develop a web service in Visual Studio .NET or compile it with the command-line compiler.

Chapter 3, *Consuming Web Services*

Web service clients can be written in any language that provides an XML parser and any platform that supports the HTTP protocol. However, you’re likely to find that the tools included with the .NET Framework make .NET applications the best choice. In this chapter, you’ll learn how to generate a proxy class that handles web service communication automatically and allows any .NET application—including Windows Forms, ASP.NET pages, or even console utilities—to access a web service without worrying about the low-level details.

Chapter 4, *Working with Data Types*

Sharing data between different languages, platforms, and operating systems opens up many opportunities and challenges. This chapter provides an advanced exploration of web service data types and how they are encoded.
You’ll learn what can be sent as a web method parameter or return value and how to work with arrays, DataSets, XML nodes, and custom structures. We’ll also take an in-depth look at .NET’s XML serialization architecture and you’ll learn about custom data shaping.

Chapter 5, Managing State
The debate over the advantages of stateless versus stateful programs is alive and well with .NET web services. This chapter examines why web services won’t support property procedures and explores the hard realities that cause many state-maintaining web service classes to fail. We’ll consider state options on the client and server side, from the ASP.NET state service to custom cookies. We’ll also look at how you can use tokens to track your own lightweight sessions and combine them with SOAP headers to provide an elegant solution for cases in which basic information must be retained.

Chapter 6, Asynchronous Services
.NET makes web methods behave like ordinary local functions, which are synchronous by default and wait for a response before allowing execution to continue. Though this is the default behavior, there’s no reason to let it dictate how you use web services in a mature application. This chapter shows how you can remove this requirement, unshackle your code, and use common optimization patterns to make your clients more efficient. These tricks include invoking multiple methods at once, using callbacks and event notifications, and creating multithreaded clients. We’ll also look at how to create a web method that creates an asynchronous component, sets it to work, and returns immediately. This allows some advanced tricks, like automatic progress tracking and batch processing.

Chapter 7, Caching and Profiling
In order to create a successful web service, you need to code with performance in mind. This chapter shows the best practices for using output caching to speed up access and data caching to replace or supplement state management. We’ll also look at how you can use performance counters and profile .NET web services. This allows you to study the results of attempted performance enhancements before a web service goes “live.”

Chapter 8, Debugging, Tracing, and Logging
.NET web services may mimic traditional classes from the programmer’s point of view, but they can’t throw ordinary .NET exceptions. This chapter explores how you can take control of errors and notify the client with special SOAP exceptions. You’ll also learn how to use Visual Studio .NET’s integrated debugger, with its celebrated single-step execution with multi-project web service solutions, and how to use .NET Framework services for automatic logging and tracing. This chapter ends with an introduction to SOAP extensions and sample code that shows you how to use a custom extension to peer beneath .NET’s object layer and examine or log raw SOAP messages en route.
Chapter 9, *Security and Authentication*

Web services standards don’t currently provide any integrated security mechanism, but you have countless options, from transport-level security to IIS authentication to ASP.NET’s own authentication services. In this chapter, you’ll learn how you can secure your web service—and more importantly, what techniques work best.

Chapter 10, *Publishing and Discovery*

Now that you’ve created the perfect web service, how do you get the word out? This chapter looks at the technologies that allow clients to track down the web services they need, including everything from simple DISCO files to online web search engines and the UDDI registry that lets businesses share details about their internal processes. We’ll consider static and dynamic discovery with DISCO files, which aggregate web services into simple groups. We’ll also examine the UDDI registry, how to use it to register a business and add services, and consider whether it’s ready for prime time.

Chapter 11, *Interoperability*

This chapter focuses on service interoperability. It discusses interoperability problems that can occur between SOAP implementations and what’s currently being done to iron out discrepancies in SOAP implementations. It also covers some simple steps you can take to make sure your .NET web services are accessible by other implementations.

Appendix A, *Namespace Quick Reference*

This appendix provides an alphabetically organized reference to the key classes of the `System.Web.Services` namespace.

Appendix B, *Web Service Technologies*

This appendix offers a list by category of some of the most frequently mentioned web service standards and technologies, and where to look for additional information.

**Conventions Used in This Book**

The following font conventions are used in this book:

*Italic* is used for:

- Pathnames, file directories, and filenames
- Internet addresses, such as domain names and URLs
- New terms where they are defined

*Constant width* is used for:

- Code samples and commands
- Names and keywords in .NET programs, including method names, variable names, names, and classes.

xv | Preface
You should pay special attention to notes set apart from the text with the following icons:

- This is a tip, suggestion, or a general note. It contains useful supplementary information about the topic at hand.
- This indicates a warning or caution. It will help you solve and avoid annoying problems.

Comments and Questions

Please address comments and questions concerning this book to the publisher:

O’Reilly & Associates, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
1-800-998-9938 (in the United States or Canada)
1-707-829-0515 (international/local)
1-707-829-0104 (fax)

There is a web site for the book, which lists examples, errata, or any additional information. You can access this page at:

http://www.oreilly.com/catalog/prognetws/

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about books, conferences, Resource Centers, and the O’Reilly Network, see the O’Reilly web site:

http://www.oreilly.com

Acknowledgments

Writing a book about any part of the .NET Framework is a challenge. Bookstore shelves are swamped with .NET titles, but few reflect the real issues and challenges practicing developers face as they tackle the new platform. Digging into the low-level details is a task that’s ideally suited for a publisher like O’Reilly, where everyone prizes technical integrity. This book couldn’t have been written without the help of many folks at O’Reilly, including John Osborn, who saw our complementary styles and brought us together to cowrite the book, and Nancy Kotary, who kept the book on track through countless technical and editorial revision cycles (even if it required dozens of emails a day). We also owe a heartfelt thanks to the technical reviewers.
who submitted timely comments and tested our code creations, including Martin Gudgin, Chris Jones, and B. Robert Helm, as well as Brian Jepson, who, although not directly tied to this project, can always be counted on to provide helpful feedback or track down a Microsoft contact. Without the support of these people and many more at O’Reilly, the book would never have been written.

**Alex**

Several people deserve my sincere and deep-felt gratitude for their contributions and/or unwavering support during the writing of this book. In addition to writing half of this book, my coauthor Matthew contributed his technical expertise, attention to detail, and a continual enthusiasm for the subject matter throughout the course of this year-long project. Matthew, working with you has been a great experience, and I owe you my thanks for all of your hard work. I’d also like to thank the U-inspire, Inc./Boston Technical team for their understanding in allowing me the time I needed to write. I owe special thanks to Matt from Boston Technical for all of the organizational and technical feedback he provided. Last, this book would not have been possible without the continuing love, understanding, and support of my wife. I love you, Tracy.

**Matthew**

First, I’d like to thank Alex, who not only wrote half the chapters but also offered technical, organizational, and philosophical feedback on my work from its initial conception to the final product. It’s been a pleasure working with you, and I’m sure the final product is more than the sum of its two parts. Next, I’d like to recognize the contribution of a few people who were key to my web services work: Nora, Razia, Hamid, Paul. Finally, this book would never have been written without the endless support of my loving wife.
Web services are about sharing functionality across the boundaries of devices, networks, operating systems, and programming languages.

In this chapter, we’ll consider four topics:

• Web services—what they are and where they fit into software development
• The technologies such as DCOM and IIOP that were the precursors to .NET web services and their limitations
• The underlying technologies and standards that support .NET web services, including SOAP, HTTP, WSDL, and UDDI
• Competing web service implementations and how they might work in harmony with .NET

These topics are covered in greater detail in later chapters.

**Why Web Services?**

Today, you can access the Internet via many different devices, connecting to a wide range of functionality beyond that available on the devices themselves. The early age of static web pages has given way to sophisticated applications for e-commerce, stock trading, team management, and a whole lot more.

The defining characteristic of the Internet is that information is transmitted (for the most part) in a pattern of requests and responses using open protocols like the pervasive HTTP. Currently, most of this information takes the form of HTML markup tags that build basic user interfaces in the form of web pages, which are viewed in a browser. Although platforms like ASP.NET use it to create applications with a wide range of features, HTML only scratches the surface of what enterprising programmers can really do with the Internet.
Web-Enabled Applications

Recently, the worlds of web and desktop application development have moved closer to each other. Desktop interfaces, like those of Windows XP, have started to resemble Internet browsers. Internet functionality has entered ordinary desktop applications, and Windows applications now interact with web servers over HTTP. For example, Microsoft Money can automatically download banking information; the Windows operating system can automatically notify you when operating system upgrades are available; and Visual Studio .NET allows you to search the MSDN library without leaving the development environment. You can even read Hotmail messages inside Microsoft Outlook and monitor eBay auctions with third-party desktop tools.

While the addition of such features is clearly an innovation, most are handcrafted, proprietary solutions. In order to add web-aware features to an application today, you must develop an infrastructure from scratch. For example, Microsoft Money needs code to interact with multiple web sources, each with its own protocols, in order to gather data from more than one bank for a user. Third-party eBay clients rely on screen scraping that will break whenever eBay updates its web site user interface. There is no web equivalent for prepackaged, reusable COM components.

Web applications are equally hobbled. Relatively clumsy methods like links, frames, and screen scraping must be used to integrate the functionality of different sites. The problem is that these applications are “monolithic”: they exist as all-in-one packages, and there is no easy way to decouple a user interface from its functionality, to provide, for example, a stock quote or a package delivery lookup, without forcing the user to go through a browser.

This is where web services and .NET come into the picture. Web services protocols specify a framework for invoking functions over the Internet. They are based on open standards and adopt a loosely coupled and extensible design that can be used for the current generation of web-enabled applications, and much more. The .NET Framework is a highly optimized platform and set of tools for deploying web services.

What Is a Web Service?

A web service is an application or block of executable code that is hosted on a web server and whose functions are exposed through standard XML protocols. .NET Framework services make it easy to find this code, invoke it, and retrieve a result. In fact, calling a .NET web service is just as easy as calling a local function.

A .NET web service is not an object! At least, not in the traditional sense. A web method is essentially independent, stateless, and atomic. A web service is more like a library of functions in a DLL than a true object-oriented abstraction. This simplicity is a large part of the appeal of web services. Because web services aren’t tightly bound to a specific technology for security, state management, or transport, they can
be used in almost any development scenario. This is completely different than earlier technologies like COM and CORBA, as you’ll see later in this chapter.

The Roles of Web Services

There’s still some confusion about who actually uses a web service. Many have mistakenly assumed that web services are somehow like application service providers, which “rent” their software by the month and allow subscribers to use it over the Internet. This is far from either current reality or the .NET vision. Web services might enable some new business models but do not endorse or constitute any particular arrangement.

Instead, web services are a way to share programming functionality. You can think of them as “COM for the Web,” even though the underlying technology is dramatically different.

Some of the ways you can use web services include:

To enable business-to-business transactions or to connect the internal systems of separate companies

This is perhaps the most common way that web services will be used for the next year or two. Web services can enable document and knowledge sharing or the integration of related services. A web service could be used, for example, to help an e-commerce company interacts automatically with a shipping company to fill orders. In these cases, the web service consumer is probably software used internally in an organization.

As prebuilt modules for developers

For example, a third-party developer could create a web service for authentication to be used by ASP.NET sites. If you chose to use this service, you might pay a monthly subscription fee based on usage, but the process would be completely transparent to the end user, who would see the functionality is an integrated part of your application. Such prebuilt components can easily be consumed by web, desktop, and mobile applications.

As value-added product features for client applications

For example, since Microsoft wants IT departments to deploy the Windows operating system, the company has an incentive to build in technology that lets system administrators perform remote administration using web services. A bank that wants you to open an investment account has an incentive to provide a web service for downloading transaction information that can be used at no charge by financial software like Quicken. While end users don’t directly consume the service, its availability in Quicken might motivate them to open an account at the bank that provides it.
About the Authors

Alex Ferrara is CTO of Boston Technical, a Boston-based IT consulting firm that specializes in custom application development and systems integration using Microsoft technologies. During his career as a consultant, Alex has had the opportunity to both manage and implement a broad range of technology initiatives for companies including Citigroup, Inc., Titleist, Inc., and Schering-Plough, Inc. He has worked with educational organizations such as Kaplan, Inc. to develop Microsoft Certification courses and has served as a professor in adjunct at Northeastern University. Alex graduated from the University of Pennsylvania with a degree in electrical and computer engineering. He currently lives in New York City with his wife, and is pursuing an MBA at Columbia Business School.

Matthew MacDonald is an author, educator, and MCSD developer. He has written several books about programming with .NET, including The Book of VB .NET (No Starch) and ASP.NET: The Complete Reference (Osborne McGraw-Hill). He has also been a contributor to several O’Reilly titles, including C# in a Nutshell, ASP.NET in a Nutshell, and ADO.NET in a Nutshell. In a dimly remembered past life, he studied English literature and theoretical physics.

Colophon

Our look is the result of reader comments, our own experimentation, and feedback from distribution channels. Distinctive covers complement our distinctive approach to technical topics, breathing personality and life into potentially dry subjects.

The animal on the cover of Programming .NET Web Services is a boatbill heron. The boatbill heron is native to Central and South America. It is distinguished by its large, wide, flat beak, which is shaped like an upside-down boat. It is a nocturnal creature and does most of its hunting for food in swamplands, but lives in trees. It’s diet consists mainly of small fish, snakes, and worms. The boatbill heron is shorter and chunkier than it’s heron relatives, and is usually brown or gray, with black markings near the top of its head.

Mary Brady was the production editor and proofreader for Programming .NET Web Services. Norma Emory was the copyeditor. Claire Cloutier and Mary Anne Weeks Mayo provided quality control. Johanna Van Hoose Dinse wrote the index. Production support was provided by Derek Di Matteo.

Ellie Volckhausen designed the cover of this book, based on a series design by Edie Freedman. The cover image is a 19th-century engraving from the Riverside Natural History, Volume IV, Birds. Emma Colby produced the cover layout with QuarkXPress 4.1 using Adobe’s ITC Garamond font.

David Futato designed the interior layout. This book was converted to FrameMaker 5.5.6 with a format conversion tool created by Erik Ray, Jason McIntosh, Neil Walls, and Mike Sierra that uses Perl and XML technologies. The text font is Linotype
Birka; the heading font is Adobe Myriad Condensed; and the code font is Lucas-
Font’s TheSans Mono Condensed. The illustrations that appear in the book were
produced by Robert Romano and Jessamyn Read using Macromedia FreeHand 9 and
Adobe Photoshop 6. The tip and warning icons were drawn by Christopher Bing.
This colophon was written by Mary Brady.