Applied SOAP: Implementing .NET XML Web Services
By Kenn Scribner, Mark C. Stiver

Copyright
About the Authors
About the Technical Editor
Acknowledgments
Author's Note
Tell Us What You Think!

Introduction
Applying Web Services to SOAP
Today's Enterprise Applications
Who Should Read This Book?
Contents of This Book
Version Issues

part I. Foundations of Web Services
Chapter 1. Web Service Fundamentals
  What Are Web Services?
  The Road to Web Services
  Uses for Web Services
  Web Service Properties
  Creating a Web Service in Visual Studio .NET
  Interface Design Tips
  Summary

Chapter 2. .NET Architecture and Web Services Components
  Motivation for Creating .NET
  The .NET Framework
  The Common Language Runtime
  Web Services
  Summary

Chapter 3. Web Services and XML
  XML as a Wire Representation
  Querying XML Elements Using XPath
  Essential XML
  URIs and XML Namespaces
  XML Schemas
  Identifying XML Elements Using XLink
  XML Transformations
Chapter 4. .NET Web Services and SOAP
   Why Is SOAP Needed?
   The SOAP XML Object Model
   The SOAP Envelope
   The SOAP Header
   The SOAP Body
   .NET SOAP Classes
   Summary

Chapter 5. Web Service Description and Discovery
   Web Service Description Language
   Universal Description, Discovery, and Integration
   Summary

part II. Implementing Web Services
Chapter 6. Web Services in ASP.NET
   Web Service Processing in .NET
   Web Services and Visual Studio .NET
   Controlling the SOAP Serialization Format
   Adding SOAP Headers
   Adding a SOAP Extension
   Errors and the SOAP Fault
   Web Service State Management
   Debugging and Deployment
   Web Services and Best Practices
   Summary

Chapter 7. Consuming .NET Web Services
   Visual Studio .NET Web Service Support
   Consuming Web Services
   SOAP Headers
   Intercepting and Modifying SOAP Packets
   More Deployment and Debugging
   Summary

part III. More Advanced Web Services
Chapter 8. .NET Remoting
   .NET Remoting Architecture
   Remotable Objects
   .NET Remoting Example
   Summary

Chapter 9. Extreme Web Services
   Embedded XML
   SOAP and Attachments
   Transactions
   Debugging and Web Services
   Web Service Documentation
   Summary
Chapter 10. .NET and Web Service Security
   Security Terms and Concepts
   Application-Level Security Versus System-Level Security
   Web Services and Security
   Breadth of Web Service Security
   .NET Security
   Summary

part IV. Appendixes
appendix A. Example .NET Web Service
   Tip of the Day Web Service in Visual Basic .NET
   finger Web Service in C"v2" colspan="18">
appendix B. Using ATL Server to Create Web Services
   ATL Server Architecture
   Example ATL Server Web Service

appendix C. XML Protocol and SOAP
   The Birth of XML Protocol
   The XMLP Abstract Model
   SOAP v1.2
   XMLP, SOAP, and the Future

appendix D. .NET Web Service Resources
   XML General
   General .NET Information
   General Web Service Information
   SOAP/XML Protocol
   Remoting
   UDDI
   WSDL
   Transactions
   Tools
   Security
   ebXML
   Sample Web Service

Index

Applied SOAP: Implementing .NET XML Web Services
By Kenn Scribner, Mark C. Stiver

Publisher: Sams Publishing
Pub Date: October 26, 2001
ISBN: 0672321114
Pages: 432
In this book you'll learn how to apply practical solutions to building XML Web Services using the .NET Framework and Visual Studio.NET that put the hot XML-based communication standard Simple Object Access Protocol (SOAP) through its paces. With the proliferation of design methodologies, programming languages, operating systems, and network architectures, achieving the level of interoperability required by Internet applications can be a significant challenge. Applied SOAP is an essential tool in mastering that challenge for intermediate to advanced level developers who are new to Web Services. This book provides a firm foundation for developers, including a view of Web Services from an architectural and design standpoint, as well as information regarding advanced Web Services and SOAP topics, such as the W3C's work on the XML Protocol (XMLP).

Copyright

Copyright © 2002 by Sams Publishing

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

Library of Congress Catalog Card Number: 00-110536

Printed in the United States of America

First Printing: October 2001

03 02 01 00 4 3 2 1

Trademarks

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


Java is a registered trademark of Sun Microsystems, Inc.

Websphere is a registered trademark of IBM, Inc.
Warning and Disclaimer

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

Credits

Associate Publisher
Linda Engelman
Acquisitions Editor
Linda Scharp
Development Editor
Laurie McGuire
Managing Editor
Charlotte Clapp
Project Editor
Leah Kirkpatrick
Copy Editor
Krista Hansing
Indexer
Larry Sweazy
Proofreader
Wendy Ott
Technical Editor
Scott Seely
Team Coordinator
Lynne Williams
Media Developer
Dan Scherf
Interior Designer
Dedication

To my mom, who hasn't had a book dedicated in her honor, and to my lovely wife, Judi, and wonderful children Aaron and Katie, who have had three (so far). You make my life rich and complete.

Kenn Scribner

To my beautiful wife and best friend, Donna, and to my three beautiful children Brendan, Nicholas, and Sydney. My greatest reward as a husband and father is the love that they give so freely.

Mark C. Stiver

About the Authors

Kenn Scribner is a full-time consultant specializing in distributed applications, COM, Windows programming, and systems architecture. Kenn also instructs for Wintellect (www.wintellect.com), where he wrote both the XML for Windows Programmers and Web Services courses. He has written or contributed to a number of books, including Understanding SOAP (Sams Publishing), Sams Teach Yourself ATL Programming in 21 Days (Sams Publishing), MFC Programming with Visual C++ Unleashed (Sams Publishing), Sams Teach Yourself DirectX 7 in 21 Days (Sams Publishing), and Professional MFC (Wrox Press).

Mark C. Stiver is a consulting software engineer with the largest news and business online information service, LexisNexis. Mark has over 12 years of distributed application experience in commercial, industrial, and military software development and is the co-author of Understanding SOAP (Sams Publishing). Mark is currently a major contributor to the architecture, design, and development of Web Services for large-scale data warehouse systems.

About the Technical Editor

Scott Seely is a member of the MSDN Architectural Samples team. He is co-author of the "At Your Service" MSDN Voices column with Matt Powell. Besides his work at Microsoft, he has published two books through Prentice Hall: SOAP: Cross Platform Web Service Development Using XML and Windows Shell Programming. He wrote and maintains a small C++ based SOAP library (http://www.scottseely.com/soap.htm), published under the LGPL.
Acknowledgments

I think the best part about writing a book is that you have the opportunity to formally thank everyone who dedicated their time to your effort for little or no reason other than to selflessly help you. It takes an amazing number of people to bring this material to you—more than just myself, a computer, and a word processor.

First and foremost, I would like to thank my wonderful wife, Judi, for her patience and support as I wrote on into the night. For far too many evenings I sat and worked on this material, and you can bet we'll catch up on movies and dinners out. To my children I offer my thanks for their support and smiling little faces, even as I tell them I couldn't toss the baseball or kick the soccer ball at that precise moment. We'll be throwing and kicking a lot more now too! I would also like to extend my deepest thanks to Mark and his family, for without their dedication and assistance you would be holding a book half as thick and only one quarter as informative. Thank you, Mark and Donna, for standing with me for "just one more."

I'd like to thank my new friend, Scott Seely, for not only volunteering to help us with this work but also for his friendship and advice. If you happen to find any technical errors in this material, the fault is mine alone and not Scott's (he probably told me to fix it!). This is a far better book because of his assistance with the technical end of .NET XML Web Services.

As I progress through my career, I'm finding myself with more and more to do at any given time. I'd like to thank Dave Pledger and Pat Caruso for their kind offers to give me time to complete this effort. This was no small contribution on their part because of the incredible workload we've sustained to move a new distributed automotive application into the marketplace.

We all should extend a huge "thank you!" to Linda Engelman for spearheading this project once we got started. I fully understand the pressures she is under to bring you the finest .NET information available. Of course, I can't forget to thank Linda Scharp for kicking this project off! We wouldn't be here today without her efforts. But if you want to thank the really hard-working crew, you surely have to thank Laurie McGuire, Charlotte Clapp, Leah Kirkpatrick, Krista Hansing, Larry Sweazy, and Wendy Ott for taking the raw materials I gave them (and they were raw, too) and turning them into the fine production you see here. Thanks to you all. Of course, I know where the real work is done, and that's in the back office, so to all of you there at Pearson that helped this book get out the door, thank you!

I simply must say a hearty "thank you!" to my Wintellect cohorts who were always there to address an issue or two. I gathered many of my thoughts and the organization for this book through the .NET Web Services course they allowed me to author. With any luck I can return the favor by providing knowledge and information, and perhaps even help one of them someday.

And of course there are a number of folks at Microsoft that held their laughter as I asked dozens of questions by e-mail. If I couldn't figure something out, I wrote them. If I couldn't get something to work, I wrote them. But it was nice having them on the team! I'd specifically like to thank Keith Ballinger who not only helped me but also drove the .NET Web Service architecture you'll see in this book. That was a monumental undertaking on his part and one I believe exceptionally well executed. Keith, I will send you that chili I owe you (and I'm hopeful you'll find a cool .NET polo somewhere in a back closet!).

Finally, I buy books, just as you do. But this time I'm on the other side, and for picking up this book and rummaging though its contents I thank you, the reader. I work long and hard hours for my paycheck, and I've met precious few developers who don't. So I know that when you lay down the money to purchase this book, it's hard won money. I truly hope you'll find this material interesting and useful. If you have questions, I'm available at kenn@endurasoft.com. If you like what you read, I thank you. If not, I still thank you and hope you'll drop me a note so I can try to address the issue.
Kenn Scribner

I want to thank all of the people that made this book possible, starting with my wife, Donna. The sacrifices that she made enabled me to focus on what seemed like an endless labor of coding and writing. I also want to thank my children for patiently waiting for their father to re-emerge from the abyss that we call my den. A huge thank you goes to Kenn Scribner, a great teacher and an even better friend— as always, it's a pleasure working with you. I specifically want to thank the XML Gateway team: Brian Bailar, Brent Ballard, Carol Becher, Nisha Bhatt, Geary Biggs, Caleb Deupree, Doug Heitkamp, Bob Hodgeman, Nick Hughes, Dean Myers, Brian Rambacher, Joe Riess, and Les Tolkkinen. I'd like to give special thanks to Ron Meyer and Chris Cardiff, for leading our team on such a great new adventure. It's unfortunate that I don't have more room on this page to individually thank everyone at LexisNexis who has supported me over the years. I've been extremely fortunate to be surrounded by such a great team of professionals.

I want to thank Microsoft and the .NET development teams for building such a great new platform—the fun is just beginning. I'd also like to thank our technical editor, Scott Seely, for his outstanding contributions to this work. Scott kept us on our toes from day one—there's no question that his diligence has produced a higher quality book.

Thanks to Linda Engelman, Laurie McGuire, Leah Kirkpatrick, Krista Hansing, and the rest of the staff at Sams Publishing. These people worked hard to mold a mere concept into something that can be placed on a shelf.

As always, I want to thank you, the reader. In a myriad of technical books, you've chosen to purchase ours. My hope is that this book will answer a question, provoke a thought, or simply provide some insight. If you have questions or comments, please contact me at mstiver@yahoo.com. Once again, thank you!

Mark C. Stiver

Author's Note

This book is about the Web Service, and more specifically about the Web Service as implemented by .NET. We tell you how to use .NET to create useful and interesting Web Services. We touch on why Web Services exist and what some alternative architectures might be that incorporate Web Services tomorrow where none exist today. But I wanted to draw your attention to a specific point before you dive into the technical material.

Here is that point. Web Services are about integration. Integration is about collaboration. Collaboration is about strategic partnerships. And strategic partnerships drive the industry as a whole. I'm a techno-wienie, and chances are you are also. But even a techno-wienie such as myself can see that the Web Service may be one of those rare technologies that can change the nature of our business (Bill Gates knows this also, if you happened to see his open letter in the September issue of "MSDN Magazine"). As important as we might believe .NET to be, it's the Web Service that will marry entire industries together as we bring remote corporations into the nth tier of our business layer.

Whether you buy this book, or another, you owe it to yourself to learn what Web Services are all about. Just as the Internet exploded on to the scene in the early 1990s, so will Web Services explode in the early 2000s. I can't say you'll make a few hundred million dollars by creating a startup technology firm, but I can say that those who know and understand Web Services will probably do well in the job market in the not-so-distant future. And maybe some of you will make a few hundred million (just remember your old pals Mark and Kenn). This is a significant technology, and it is yours to use and profit from. We wish you the best of luck as we enter this new world.
Finally, I owe you a technical side note. As I write this, the summer of 2001 draws to a close. At this time, we're all using Beta 2 of the .NET Framework and Visual Studio .NET. The material contained within this book was drawn from experience using the .NET Framework, various magazine articles, the online documentation for .NET, and through many personal interviews and requests for information. We anticipate that very little will change between the second beta and the actual release of .NET to the general public, but there is no guarantee. If you happen to be reading this after the official release of .NET and you find things don't work as we claim, rest assured we're aware of the situation and will correct the book if given an opportunity. Drop me a note (kenn@endurasoft.com) and I will be happy to help you find the correct solution. It is my intention to maintain current code at http://www.endurasoft.com/netws.htm, so drop by my site for some code and updates if you have the chance.

Tell Us What You Think!

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

As an associate publisher for Sams, I welcome your comments. You can fax, e-mail, or write me directly to let me know what you did or didn't like about this book as well as what we can do to make our books stronger.

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

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

Introduction

This book is about a new approach to building applications for the Internet—the technology is called Web Services.

Quite a lot of changes have taken place where the Internet is concerned, especially with regard to how information and services can be shared. If you can disregard the typical marketing hype for a short time, you will see that Web Services are really just a new way for applications to share information using existing and proven Internet technologies.

But it doesn't stop with the Internet. Web Services also offer the opportunity for services to be shared between applications on the same machine, within an intranet, or even between devices.

Web Services are all about interoperability, and XML is one of the fundamental enabling technologies behind them. However, XML alone isn't enough—the Simple Object Access Protocol (SOAP) defines a common way for systems to use XML as a communication protocol.
Applying Web Services to SOAP

In our last book, *Understanding SOAP: The Authoritative Solution*, we discussed SOAP from a distributed application and Remote Procedure Call (RPC) perspective, with a strong emphasis on protocol-specific features. Although not everyone will want to learn the ins and outs of SOAP, understanding the implications of the decisions that you make about the protocol will definitely make you a better Web Service developer.

Since the release of *Understanding SOAP*, many new technologies, such as WSDL and UDDI, have emerged to extend the capabilities of SOAP-enabled systems. Several new SOAP implementations have been released as well (the last unofficial count suggested that there are roughly 68 implementations), and the industry is seeing Web Services gain momentum in real-world applications.

Because of many of these changes, we've taken a slightly different tact in this book: We've decided to look at SOAP in a larger context--properly applying SOAP to Web Services. Although SOAP is not the only approach to implementing Web Services, alternatives such as directly using HTTP GET and POST simply aren't as flexible and thus won't be emphasized.

This book devotes more attention to document-oriented messaging than to RPC. Using SOAP as an RPC mechanism is extremely helpful when you're trying to preserve object identity and other features typically supported by current RPC protocols. The fact is, most of the existing implementations are based on the RPC approach, but this is quickly changing. In the business-to-business (B2B) arena, the document-based solution is more direct and much easier to describe in schema languages. We will try to shed some light on the pros and cons of using RPC versus literal XML documents because this will become an interoperability issue as time goes on.

As I've already mentioned, you no longer need to build your own SOAP implementations (I can already hear your cries of disappointment). As you might expect, there are plenty of good implementations for a variety of different programming languages and platforms. The mainstream offerings include Apache's Java SOAP toolkit (originally IBM's SOAP4J), Microsoft's SOAP Toolkit, and several different libraries for Perl, C++, and others.

So why did we choose to write about Microsoft's .NET Framework? Our initial response is because .NET has created an entirely new runtime that's designed specifically for Internet application development. But somewhere deep inside, we know that part of the reason is because .NET is just plain cool. It's especially interesting to those of us who have been doing Windows development on and off for many years. Windows development has had its share of blemishes, not uncommon for a system that has undergone many new face-lifts and architectural changes. .NET represents the playground equivalent of a do-over. The developers at Microsoft took a long look at what is working in the industry and improved upon it. The end result is a framework for building applications that simplifies the programming model while still maintaining backward compatibility with existing architectures such as COM.

We've been around long enough to know that there's no panacea in software development. To be a good software developer, it's important to be pragmatic. You need to question why things work the way they do, you need to be critical about the technologies you choose, and you have to be willing to try new things. It's possible that all of these played a part in your decision to pick up and read this book. They definitely contributed to our decision to write this book about Web Services and .NET!

Overall, this book has three main themes. First, it approaches Web Services from an architectural and design standpoint to analyze the issues that must be addressed in this new paradigm, specifically within the .NET Framework. Second, it shows you how to implement Web Services using the .NET Framework and the new Visual Studio .NET tools. Finally, more advanced Web Services and SOAP topics are discussed, including information regarding the W3C's work on the XML Protocol (XMLP).
Software development is a constantly changing art, and programmers have the daunting responsibility of staying up-to-date with the latest trends in technology. The purpose of this book is to teach you how to apply practical solutions to building Web Services using the .NET Framework. The greatest advantage to Web Services is that it applies to so many problem spaces. These problem spaces include complex B2B system integration, networked devices, distributed development, and many others. However, with this promise of interoperability and simplicity comes a great deal of expectations.

Many advancements in our field have made software development a much easier task--and, at times, a much harder one. Design methodologies, programming languages, operating systems, network protocols, distributed architectures--all of these and others impact the way we build software. Unfortunately, all of this technology leads to a staggering number of software incompatibilities that become even more apparent when cast upon the Internet backdrop.

The industry has tried to address these incompatibilities in a number of ways. Java addresses the problem from a programming language perspective, allowing applications to be written once and deployed to any number of platforms that support a Java Virtual Machine. Through the use of Java's Remote Method Invocation (RMI) technology, applications can more easily operate in a distributed environment. When considering Java and RMI however, to guarantee interoperability, you are responsible for ensuring that both ends of the connection are running Java. In a controlled environment this could be a viable solution, but the Internet is far from a controlled environment.

Both CORBA and DCOM (DCE/RPC) address the interoperability problem from a networking perspective. Given a common way to encode information into network packets (using Network Data Representation), programming languages become less of an issue. Instead, maintaining compatibility with serialization methods becomes the concern. This usually means that to build compatible components, you are forced to use one particular vendor's implementation. Although the vendors would love this, it would be impossible to get everyone on the Internet to agree to use just one vendor's software.

Web Services offer a layer of abstraction that hides many of the implementation details that so many developers can't seem to disregard. Take programming languages, for example. Programmers tend to gravitate toward one particular language and discount other (possibly competing) languages. Sometimes these observations are based on experience, and sometimes the reasons are purely religious in nature, such as the animal desire to see semicolons and curly braces in code. Regardless of the basis for these feelings, this choice is a fact of life for developers and will not likely ever go away. Certainly, the .NET Framework goes a long way toward marrying diverse programming languages into a common environment--but this is true only on platforms that support .NET. Web Services take this one step further by breaking down the platform barriers to allow heterogeneous systems to interact.

The Internet has introduced technologies that are a sort of common denominator. Interoperability is the driving force behind most of the successful Internet technologies. Without interoperability, a particular technology has little chance of being adopted. Without adoption, even the most interesting technologies become nothing more than graduate research projects. The reason that the Internet has been so successful is the increasingly popular movement toward standardization. It started with the adoption of TCP/IP, the cornerstone of the Internet communication protocols, and has continued with the acceptance of HTTP, XML, and, more recently, SOAP.
In actuality, SOAP isn't considered a standard—at least, not in terms of the W3C. But it is considered industry accepted, as can be seen by the enormous development community interest. The fact is, SOAP is currently under the looking glass of the W3C's XML Protocol Activity. In this forum, W3C members get the opportunity to shape the technology into something that addresses the needs of the industry.

Today's Web applications are expected to handle thousands of transactions per minute to solve a wide array of problems. As we build upon an architecture in which applications interact with other applications, the frequency and size of transactions will most definitely grow. This will cause us to think about the Internet in entirely new terms, eventually leading to a metamorphosis in the systems that make up the network. Concerns about reliability, availability, performance, and similar properties will need to be addressed under this new development model.

The Search for Solutions

There's still a lot to learn about distributed computing, and Web Services will force us to extend our knowledge in the Internet space. This time around, however, we'll be forced to think in terms of standards that will allow all kinds of systems to participate.

The most important concept behind Web Services is that software functionality can be shared regardless of the implementation details behind these services. This level of integration is accomplished through loosely coupled services that provide a flexible and responsive architecture for building the next generation of applications.

Who Should Read This Book?

This book is for the intermediate to advanced developer who is new to Web Services. It will show you how to use the .NET Framework and Visual Studio .NET to construct Web Services. It also will help you understand common pitfalls and issues that surround this new architecture. Software engineers, programmers, and Web developers alike will find many of the concepts applicable to their daily development lives when building these systems. As always, our goal is to save you development time by presenting practical information through discussion, examples, and source code.

The examples in this book mostly contain Visual C# code, with an occasional Visual Basic snippet. We chose C# because of its close relationship to Java and C++ programming, which should meet the needs of a fairly wide audience.

The majority of this book is dedicated to the SOAP protocol as it has been implemented within .NET. Although some introductory SOAP information is provided in Chapter 4, "NET Web Services and SOAP," we assume that most readers are already familiar with the protocol and are simply looking for new insights when applying SOAP to .NET applications.

Contents of This Book

This book is divided into three sections.

Section 1, "Foundations of Web Services," describes what Web Services are and how the .NET architecture applies to this paradigm.
Chapter 1, "Web Service Fundamentals," provides a brief explanation of Web Services and tells why they are the future of Web application development.

Chapter 2, "NET Architecture and Web Services Components," takes you on a quick tour of the .NET architecture. It also gives you an overview of the Common Language Runtime and specific Web Service components.

Chapter 3, "Web Services and XML," shows how the .NET architecture integrates XML with the new .NET XML classes.

Chapter 4, ".NET Web Services and SOAP," explains how SOAP is used to provide a consistent serialization format, fault pattern, and general protocol for Web Services.

Chapter 5, "Web Service Description and Discovery," details the Web Service Description Language as well as the Universal Description, Discovery, and Integration concepts.

Section 2, "Implementing Web Services," walks you through the ASP.NET and Visual Studio .NET tools.

Chapter 6, "Web Services in ASP.NET," covers development from the server-side perspective.

Chapter 7, "Consuming .NET Web Services," shows the latest features provided in Microsoft's newest development environment.

Section 3, "More Advanced Web Services," discusses several issues and provides multiple options to consider when encountering these problems.

Chapter 8, "NET Remoting," covers issues such as how SOAP can be used to carry various forms of payloads, encryption, and alternate encoding formats.

Chapter 9, "Extreme Web Services," closes the book's discussion by establishing a basis for understanding authentication, authorization, entitlements, and digital signatures as they relate to Web Services.

Chapter 10, "NET and Web Service Security," overviews the options available when securing your Web Service.

In addition, this book provides reference material in the following appendixes:

- Appendix A "Example .NET Web Service"
- Appendix B "Using ATL Server to Create Web Services"
- Appendix C "XML Protocol and SOAP"
- Appendix D "NET Web Service Resources"

Version Issues

This book is based on a pre-release version (Beta 2) of Visual Studio .NET and the .NET Framework. It's possible that features of the fully released product could be inconsistent with topics covered in this book. Potential changes to look for include minor organization changes in the .NET libraries, minor parameter changes in classes, and significantly improved documentation.

Regardless of the changes to .NET, the concepts and approaches described in this book will continue to remain valid.
Chapter 1. Web Service Fundamentals

IN THIS CHAPTER

- What Are Web Services?
- The Road to Web Services
- Uses for Web Services
- Web Service Properties
- Creating a Web Service in Visual Studio .NET
- Interface Design Tips

It's pretty hard to pick up a trade magazine these days without seeing a headline about Web Services. With phrases such as "a new paradigm" being proliferated, are we really witnessing the genesis of a new technology?

Unfortunately, the answer isn't black and white—a lot depends on your perspective. Web Services can be used in a wide variety of ways, including these:

- Participating in business-to-business (B2B) transactions
- Exposing software functionality to customers
- Integrating heterogeneous platforms and programming languages
- Providing a simplified platform for product development

What Are Web Services?

Web Services can be described as any functionality that is accessible over the Internet, generally (but not necessarily) using one or more eXtensible Markup Language (XML) messages in the communications protocol. Web Services use the concept of an operation to represent the association of a request message to zero or more response messages. When these operations are combined to satisfy some particular purpose, they form an interface.

The Poor Man's Web Service

The Internet is already flooded with conventional types of Web Services, better known as Web pages. Users are expected to interact with the functionality behind the Web page through typical user-interface widgets such as forms, buttons, and so on.
We already know how to reuse Web functionality by embedding other Web pages into our own pages through frames and links. But this presentation-based approach severely limits the things that you can accomplish. If you embed another Web site within a frame of your own, you generally have no control over the colors, graphics, or other aspects of the presentation. Another problem is that any information entered by a user in the embedded page never gets back to your controlling application. In other words, you're out of the loop!

One way around this is for your application to act as a proxy for the user. Many developers have already written simple applications that navigate to a particular URL, screen-scrape the Web site's HTML for information, and use that information to build new Web content. Consider the following HTML that describes the current weather temperature:

```html
<HTML>
    <HEAD>
        <TITLE>Today's Weather</TITLE>
    </HEAD>
    <BODY>
        <P>City: <B>Los Angeles</B></P>
        <P>State: <B>California</B></P>
        <P>Temperature: <B>83</B></P>
    </BODY>
</HTML>
```

In this case, it would be fairly simple to programmatically locate the temperature value within the markup. However, over an extended period of time using this service, you can be sure that the underlying Web page will change and ultimately break your application. It would be nearly impossible to develop software that could automatically adjust to fluctuations in the type of markup, as shown:

```html
<HTML>
    <HEAD>
        <TITLE>Today's Weather</TITLE>
    </HEAD>
    <BODY>
        <TABLE WIDTH="500" CELLPADDING="10" CELLSPACING="15">
            <TR>
                <TD ALIGN="LEFT" VALIGN="MIDDLE" WIDTH="100">City</TD>
                <TD ALIGN="LEFT" VALIGN="MIDDLE" WIDTH="200">Los Angeles</TD>
            </TR>
            <TR>
                <TD ALIGN="LEFT" VALIGN="MIDDLE" WIDTH="100">State</TD>
                <TD ALIGN="LEFT" VALIGN="MIDDLE" WIDTH="200">California</TD>
            </TR>
            <TR>
                <TD ALIGN="LEFT" VALIGN="MIDDLE" WIDTH="100">Temperature</TD>
                <TD ALIGN="LEFT" VALIGN="TOP">83</TD>
                <TD ALIGN="LEFT" VALIGN="TOP"><IMG SRC="sunny.gif" WIDTH="25" HEIGHT="33"></TD>
            </TR>
        </TABLE>
    </BODY>
</HTML>
```

Now consider integrating multiple systems using this approach. Recall that in the Web Services paradigm, many systems likely could participate in some business process. Because HTML content changes at such a fast pace, you likely will not ever be able to construct a reliable integrated solution.

For example, consider one application that monitors the temperature Web site and another Web site that posts
the average speed of traffic on a nearby highway:

By relating these two axes of data, the application might be capable of determining whether there is a correlation between sunny days and fast driving. Although it might be an interesting problem to solve, the likelihood of the application working with 24x7 reliability is extremely low. The plain-and-simple fact is that relying on presentation-oriented data leads to a tightly coupled and brittle system.

The question isn't whether the concept of integrating Web content is valid; the problem lies within the information that can be obtained from a source. Without rich content markup, programs don't have much of a chance of locating pertinent information. Of course, this is where XML markup makes an important difference. Given standardized markup that describes information in a particular domain space, an application should always be capable of finding the right data.

Taking the concept of standardized markup into account, Web Services can be better defined as functionality that is accessed over the Web and that provides information in a reliable and predictable manner. In many cases, this predictability will be realized through the use of XML markup for describing information.

Although Web Services are not limited to the following technologies, you will find that a large percentage of Web Service implementations are built upon the Hypertext Transfer Protocol (HTTP), SOAP/XML as a messaging protocol, and Web Services Description Language (WSDL) as a way to describe service interfaces.

The basic idea behind Web Services isn't really new. In many ways, we are just reusing technologies that most of us have used for years. Surprisingly, many developers have already built systems using Web Service techniques, but in a very ad hoc and proprietary way. The main difference is that the industry is now supporting Web Services with standards, tools, and implementations.

First reactions about Web Services usually revolve around performance. Most people recognize that transmitting XML is not the most expeditious way for systems to communicate. So why use XML? We use XML because it provides us with a predictable way to package information that is structured, extensible, and yet still very easy to use—not something that can be said for other packaging protocols. Let's take a closer look.

XML Messages

By nature, interface-based programming enables us to build loosely coupled systems, meaning that the client and Web Service are independent of one another. This has been true in object-oriented programming for many years, within the confines of a particular programming language. Web Services reinforce loosely coupled systems by removing dependence upon a common programming language or even a common platform. This is realized through the use of XML messages, which define the operations inside a Web Service interface.

The importance of this feature is well understood by distributed application developers who have been using systems such as CORBA and DCOM. Historically, building applications on top of binary protocols and their associated runtimes results in a very tight dependency between the client and the server. This forces developers to repeatedly build and distribute new interface components (such as proxies and stubs), which is a
very tedious and error-prone process. More importantly, though, XML lets you focus on the interface semantics rather than having to worry so much about synchronizing parameter lists of remote methods.

**Syntax Versus Semantics**

Recall that syntax is the detailed representation of information. It's the way you organize instructions in a programming language or arrange tags in an XML document.

Semantics, on the other hand, refers to the meanings or concepts behind a syntactical representation. Because semantics represents information from a logical standpoint, there might be several ways to syntactically represent that information, all of which should convey the same meaning to the information consumer.

To better contrast syntax and semantics, consider the following sample XML:

```xml
<ChargeCreditCard>
  <amount>150.00</amount>
  <creditCardNumber>123456789</creditCardNumber>
  <expirationDate>2003-01-31</expirationDate>
</ChargeCreditCard>
```

In this case, the syntax is fairly simple—an XML message consisting of start and end tags, structured with a single root element and its descendants. We could have just as easily used the following text:

*Please charge $150.00 to credit card number 123456789, which expires January 31, 2003.*

Semantically, the information represented by both syntaxes allows you to bill someone's credit card, which is really what we're interested in. Obviously, the latter syntax is more pleasing to humans, and the former XML message is much more acceptable for application consumption.

SOAP uses XML to define a syntax, which makes it very easy to represent information in a structured form. However, SOAP also carries some important protocol semantics that allow SOAP processors to serialize/deserialize data, handle faults, and mandate that certain information be present in a message.

As the creator of a Web Service, you have the task of defining your own set of semantics for your Web Service. Some simple semantics might be to get a stock quote or to retrieve the time and temperature. A more advanced semantic might be to schedule a vacation, which includes reserving a hotel room, airfare, and ground transportation. Arriving at a reasonable set of semantics requires you to use standard software engineering practices such as working with domain experts.

The long-term vision of Web Services is for developers to be able to construct applications by integrating one or more units of functionality into a single service (as in the vacation example). The most significant aspect of this model is that you can incorporate distinct units of functionality from a wide variety of sources and successfully complete some larger task or business process. It is like code reuse, without the programming language compatibility problems.

To describe the interaction between Web Services and their clients, it's important to define some terminology that will be used in this context.
Web Service Terminology

The Web Services model uses several terms that help to identify the various roles in a typical Web Service scenario.

A service provider is an entity that hosts a Web Service that exposes some functionality. The service provider is responsible for defining the semantics of the service interface as well as constructing the appropriate physical representation as depicted in a Web Service description document.

The service provider can then publish the interface description to a service registry. Here, information about the provider and the service are persisted for service discovery. The service registry exposes its own set of interface semantics that allows others to create new entries, update registry information, or query for specific registry parameters. For more detailed information about service registries, refer to Chapter 5, "Web Service Description and Discovery."

At some point after service publication, a service requestor (sometimes referred to as the client) can discover the Web Service and its interface description, and bind to this service to fulfill the service requestor's needs.

As you can see from Figure 1.1, three major processes take place. First, the service provider publishes service information. Next, the service requestor finds the service in the service registry. Finally, the service requestor binds to the service to execute some functionality. This publish, find, and bind model is consistent with other networking protocols such as DNS.

Now that you are familiar with the concept of Web Services, let's take a quick tour of how Web Services came about and why.
The Road to Web Services

Most things in software engineering happen for a very good reason or, at least, we hope so. This is how concepts such as abstraction and encapsulation have become mainstream. As with any technology, we learn from past mistakes and capitalize upon our successes.

Web Services are no different. They have materialized from a variety of Web technologies that have been proven to work in the widely distributed environment of the Internet. The Internet itself has evolved over the years, spawning many new ideas and concepts that have contributed to the Web Service approach.

Waves of the Internet

Since the beginning of the Internet, many changes have come about in networking technology, security, system scalability, and many other areas of distributed computing. Overall, we believe that the Internet has succumbed to four major waves of development.

The first wave of the Internet started around the 1970s with some very important government research, specifically the Defense Advanced Research Projects Agency (DARPA). This is where Transmission Control Protocol/Internet Protocol (TCP/IP) was born. Its goal was to interconnect computer systems through a complex architecture of networks and subnetworks. Over the years, a variety of physical networks (such as Ethernet) and routing technologies evolved to the point that, in 1990, more than 200,000 computers were interconnected on the Internet.

Although Internet connectivity was one of the most significant achievements in computing, it meant very little without applications to drive it. This is where the second wave of the Internet began (roughly in the 1980s). Tools such as FTP and Telnet gained in popularity by allowing system users to remotely access other computers. Although the tools were crude, compared to today's standards, the underlying protocols that they used were quite elegant.

In the early 1990s, the Internet began to seep into more sophisticated applications and led to the dawn of the Web, which marks the third wave. Browsers and eventually Java applets allowed the general consumer to experience interconnected communities of users. Of course, where there are consumers, there are vendors. This spawned more electronic business opportunities, as evidenced by the plethora of electronic storefronts and shopping carts.

All this, of course, has brought about the fourth wave of the Internet, which is the focus of this book—Web Services. Here, the goal is for multiple diverse applications to communicate so that they can execute some task. Not only does this improve the user's experience, but it also offers the ability for you to integrate functionality at a much lower cost than developing it all yourself. From the user's standpoint, all of this is orchestrated from a single application. But behind the scenes, one or more additional applications will likely participate. The key is that the applications work while remaining oblivious to vendor-specific technologies being used by the participating services.

Looking back, each wave introduced new Internet standards that facilitated the next wave of development.

Internet Standards

In April 1969, the first Request for Comments (RFC) was published at UCLA (RFC 1), and thus began the process of sharing ideas in computing for a much greater cause. Then, in 1986, the Internet Engineering Task Force (IETF) was officially created. Its charter was (and still is) to evolve the architecture of the Internet using open contributions from the research and development community.
After inventing the Web, Tim Berners-Lee decided to create the World Wide Web Consortium (W3C) in October 1994. The W3C's purpose is to promote interoperability and open forum discussions about the Web and its protocols.

These organizations have led the way to standardization, a process that has resulted in a strong foundation for the Web Service infrastructure.

Several standards are prevalent in current Internet development. Some have existed for years, and others are relatively new, and not necessarily standards. This section summarizes these technologies and shows how they apply to Web Services.

**HTTP and SMTP**

As stated before, TCP/IP is the foundation of Internet communication protocols. However, TCP/IP without an application is a little like a car without a driver. How an application uses TCP/IP also determines the semantics of that application's protocol.

Application protocols such as HTTP and Simple Mail Transfer Protocol (SMTP) already have predefined semantics and behavior that determine how they should be used. For HTTP, the semantic implies a request/response model designed to serve Web resources such as HTML or JPG files. SMTP, on the other hand, implies a one-way request/acknowledge semantic designed to transmit text-based email messages in a fire-and-forget manner.

**NOTE**

Fire-and-forget is a military warfare term that has been overloaded for networking purposes. We use it to describe the process of sending a message: The sender does not require any acknowledgement that the recipient actually received the message.

In the context of Web Services, these application protocols are used to carry additional semantics, such as those specified by SOAP. SOAP, in turn, provides a way for you to define your application semantics that are also carried over these application protocols. This layering of semantics just reemphasizes the flexibility of Web Service protocols.

Because HTTP is the dominant protocol being used for Web Services, let's take a quick look at the two most common aspects of HTTP being exploited, the **GET** and **POST** verbs.

The following is a sample HTTP **GET** request:

```
GET /default.htm HTTP/1.1
Accept: text/*
Host: www.mcp.com
(CR) (LF)
```

In this case, the client is requesting that the www.mcp.com server return the default.htm resource. The client would like to use version 1.1 of the HTTP protocol in this transaction and is willing to accept the resource as some form of text. Notice that the message is terminated by the carriage return/line feed pair following the message.

Many times you need to provide application-specific information to the server that is not represented in the semantics of the HTTP protocol. For instance, you can pass parameters on the URL, as shown:

```
GET /default.htm?param1=value1&param2=value2 HTTP/1.1
Accept: text/*
Host: www.mcp.com
(CR) (LF)
```