



.NET Web Services Solutions

Kris Jamsa



San Francisco • London



.NET Web Services Solutions



.NET Web Services Solutions

Kris Jamsa



San Francisco • London

Associate Publisher: Joel Fugazzotto
Acquisitions Editor: Denise Santoro Lincoln
Developmental Editor: Tom Cirtin
Production Editor: Donna Crossman
Technical Editor: Acey Bunch
Copyeditor: Linda Stephenson
Compositor: Maureen Forsys, Happenstance Type-O-Rama
Graphic Illustrator: Jeff Wilson, Happenstance Type-O-Rama
Proofreaders: Emily Hsuan, Eric Lach, Laurie O'Connell,
Yariv Rabinovitch, Nancy Riddiough, Sarah Tannehill
Indexer: Ted Laux
Cover Designer: Carol Gorska/Gorska Design
Cover Photographer: Carlos Navajas, The Image Bank

Copyright © 2003 SYBEX Inc., 1151 Marina Village Parkway, Alameda, CA 94501. World rights reserved. No part of this publication may be stored in a retrieval system, transmitted, or reproduced in any way, including but not limited to photocopy, photograph, magnetic, or other record, without the prior agreement and written permission of the publisher.

Library of Congress Card Number: 2002116880

ISBN: 0-7821-4172-2

SYBEX and the SYBEX logo are either registered trademarks or trademarks of SYBEX Inc. in the United States and/or other countries.

Transcend Technique is a trademark of SYBEX Inc.

Screen reproductions produced with Collage Complete.
Collage Complete is a trademark of Inner Media Inc.

TRADEMARKS: SYBEX has attempted throughout this book to distinguish proprietary trademarks from descriptive terms by following the capitalization style used by the manufacturer.

The author and publisher have made their best efforts to prepare this book, and the content is based upon final release software whenever possible. Portions of the manuscript may be based upon pre-release versions supplied by software manufacturer(s). The author and the publisher make no representation or warranties of any kind with regard to the completeness or accuracy of the contents herein and accept no liability of any kind including but not limited to performance, merchantability, fitness for any particular purpose, or any losses or damages of any kind caused or alleged to be caused directly or indirectly from this book.

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

SOFTWARE LICENSE AGREEMENT: TERMS AND CONDITIONS

The media and/or any online materials accompanying this book that are available now or in the future contain programs and/or text files (the "Software") to be used in connection with the book. SYBEX hereby grants to you a license to use the Software, subject to the terms that follow. Your purchase, acceptance, or use of the Software will constitute your acceptance of such terms.

The Software compilation is the property of SYBEX unless otherwise indicated and is protected by copyright to SYBEX or other copyright owner(s) as indicated in the media files (the "Owner(s)"). You are hereby granted a single-user license to use the Software for your personal, noncommercial use only. You may not reproduce, sell, distribute, publish, circulate, or commercially exploit the Software, or any portion thereof, without the written consent of SYBEX and the specific copyright owner(s) of any component software included on this media.

In the event that the Software or components include specific license requirements or end-user agreements, statements of condition, disclaimers, limitations or warranties ("End-User License"), those End-User Licenses supersede the terms and conditions herein as to that particular Software component. Your purchase, acceptance, or use of the Software will constitute your acceptance of such End-User Licenses.

By purchase, use or acceptance of the Software you further agree to comply with all export laws and regulations of the United States as such laws and regulations may exist from time to time.

Reusable Code in This Book

The author(s) created reusable code in this publication expressly for reuse by readers. Sybex grants readers limited permission to reuse the code found in this publication, its accompanying CD-ROM or available for download from our website so long as the author(s) are attributed in any application containing the reusable code and the code itself is never distributed, posted online by electronic transmission, sold, or commercially exploited as a stand-alone product.

Software Support

Components of the supplemental Software and any offers associated with them may be supported by the specific Owner(s) of that material, but they are not supported by SYBEX. Information regarding any available support may be obtained from the Owner(s) using the information provided in the appropriate read.me files or listed elsewhere on the media.

Should the manufacturer(s) or other Owner(s) cease to offer support or decline to honor any offer, SYBEX bears no responsibility. This notice concerning support for the Software is provided for your information only. SYBEX is not the agent or principal of the Owner(s), and SYBEX is in no way responsible for providing any support for the Software, nor is it liable or responsible for any support provided, or not provided, by the Owner(s).

Warranty

SYBEX warrants the enclosed media to be free of physical defects for a period of ninety (90) days after purchase. The Software is not available from SYBEX in any other form or media than that enclosed herein or posted to www.sybex.com. If you discover a defect in the media during this warranty period, you may obtain a replacement of identical format at no charge by sending the defective media, postage prepaid, with proof of purchase to:

SYBEX Inc.
Product Support Department
1151 Marina Village Parkway
Alameda, CA 94501
Web: <http://www.sybex.com>

After the 90-day period, you can obtain replacement media of identical format by sending us the defective disk, proof of purchase, and a check or money order for \$10, payable to SYBEX.

Disclaimer

SYBEX makes no warranty or representation, either expressed or implied, with respect to the Software or its contents, quality, performance, merchantability, or fitness for a particular purpose. In no event will SYBEX, its distributors, or dealers be liable to you or any other party for direct, indirect, special, incidental, consequential, or other damages arising out of the use of or inability to use the Software or its contents even if advised of the possibility of such damage. In the event that the Software includes an online update feature, SYBEX further disclaims any obligation to provide this feature for any specific duration other than the initial posting.

The exclusion of implied warranties is not permitted by some states. Therefore, the above exclusion may not apply to you. This warranty provides you with specific legal rights; there may be other rights that you may have that vary from state to state. The pricing of the book with the Software by SYBEX reflects the allocation of risk and limitations on liability contained in this agreement of Terms and Conditions.

Shareware Distribution

This Software may contain various programs that are distributed as shareware. Copyright laws apply to both shareware and ordinary commercial software, and the copyright Owner(s) retains all rights. If you try a shareware program and continue using it, you are expected to register it. Individual programs differ on details of trial periods, registration, and payment. Please observe the requirements stated in appropriate files.

Copy Protection

The Software in whole or in part may or may not be copy-protected or encrypted. However, in all cases, reselling or redistributing these files without authorization is expressly forbidden except as specifically provided for by the Owner(s) therein.

*To my good friend, Dr. Guy Hodgson,
For your endless support and words
of encouragement and your willingness to
share your knowledge and expertise.*

Acknowledgments

Although only the author's name appears on a book's cover, the publication of a book requires the tireless efforts of a team of talented and dedicated individuals. Please take a moment and turn to this book's copyright page to view the members of the Sybex team that worked hard on this book.

To start, I want to thank Dr. Rodney Zaks for the opportunity to work with Sybex on this book. I have great respect for Dr. Zaks both for his ability to drive Sybex for over 25 years and for his technical and business acumen.

Next, I want to thank Denise Lincoln for her patience and support throughout this project. My special thanks also go to Tom Cirtin for his development efforts and to Acey Bunch for his technical-editing expertise. In addition, I appreciate the work of Donna Crossman and Linda Stephenson for managing and editing the book's manuscript. The efforts of the Sybex team greatly improved this book's content.

Contents at a Glance

Introduction

xxii

Part I Laying the Foundation

Chapter 1:	Taking Web Services for a Test Drive	3
Chapter 2:	Creating Your First Web Services	39
Chapter 3:	Accessing Web Services from within HTML Pages	83
Chapter 4:	Looking Behind the Scenes at Web Service Protocols	109
Chapter 5:	Looking at Key Operations	131

Part II Publishing Your .NET Web Services

Chapter 6:	Making .NET Web Services Available to Others	161
Chapter 7:	Connecting Web Services to Databases	177
Chapter 8:	Authenticating Users within .NET Services	207
Chapter 9:	Securing Communication between a Web Service and a Client	221
Chapter 10:	Extending the Lines of Communication	247
Chapter 11:	Integrating Binary Data into .NET Web Services	267

Part III Advanced Concepts

Chapter 12:	Examining Key Web Service Files	287
Chapter 13:	Unlocking Remote Access	303
Chapter 14:	Improving Web Page Performance	333
Chapter 15:	Making Money with Web Services	361
Chapter 16:	Putting It All Together	395

Index

427

Contents

Introduction

xxii

Part I	Laying the Foundation	1
Chapter 1	Taking Web Services for a Test Drive	3
	What's a Web Service?	4
	Understanding Operations That Are Well Suited for Web Services	6
	Retrieving Weather Information	7
	Looking Behind the Scenes at the FastWeather Web Service	8
	Retrieving Weather Information within a C# Program	11
	Using a Web Service 101	15
	Retrieving Stock Quotes	16
	Looking Behind the Scenes at the StockQuote Web Service	16
	Retrieving Stock Prices within a C# Program	19
	Retrieving Book Information	23
	Behind the Scenes of the Amazon Web Service	25
	Behind the Scenes of the Barnes & Noble Web Service	27
	Retrieving Caller-ID Information	29
	Behind the Scenes of the GeoPhone Web Service	29
	Retrieving Traffic Information	32
	Behind the Scenes of the CATraffic Web Service	33
	Retrieving Airport Information	35
	Behind the Scenes of the AirportInfo Web Service	35
	Where to Find Web Services on the Web	37
	Summary	38
Chapter 2	Creating Your First Web Services	39
	Getting Started with the Hello Web Service	40
	Putting the Hello Web Service to Use	42
	Changing the Hello Web Service	43

Remembering What Makes a Good Web Service	45
Putting the GreetSpecific Web Service to Use	47
Passing Parameters to a Web Service	48
Putting the DayTimeGreeting Web Service to Use	50
Returning a Non-String Value from a Web Service	51
Putting the DetermineAge Web Service to Use	53
Handling Multiple Parameters in a Web Service Method	54
Putting the SalesTax Web Service to Use	57
Changing Parameter Values within a Web Service Method	59
Putting the Swap Web Service to Use	60
Getting a Feel for Web Service Overhead	62
Passing an Array of Values to a Web Service	65
Putting the SimpleStatistics Web Service to Use	66
Changing Array Values in a Web Service	68
Putting the IncrementArray Web Service to Use	69
Returning an Array of Values from a Web Service Method	71
Putting the Presidents Web Service to Use	72
Returning a Structure from a Web Service	74
Putting the ServerInfo Web Service to Use	75
Throwing an Exception within a Web Service	76
Putting the Exception Demo Web Service to Use	78
Calling a Web Service from within a Web Service	80
Putting the NestedCall Web Service to Use	81
Summary	82
Chapter 3	
Accessing Web Services from within HTML Pages	83
Understanding and Applying HTML Behaviors	84
Downloading the Webservice.htc Behavior	84
Attaching a Web Service to an HTML Document	84
Calling a Web Service	85
Calling the Hello Web Service	86
Calling the GreetSpecific Web Service	88
Calling the Swap Web Service	90
Calling the Array Web Service	92
Calling the ServerInfo Web Service	94
Calling the Presidents Web Service	96

	Handling Web Service Errors	97
	Using a Web Service As a Proxy	99
	Using the Google Web Service	101
	Searching for Text Using the Google Search Engine	103
	Calling the Google Web Service Using the Webservice Behavior	105
	Summary	107
Chapter 4	Looking Behind the Scenes at Web Service Protocols	109
	XML 101	110
	Parsing XML	112
	XML Support for Attributes	113
	Understanding the Web Service Description Language (WSDL)	113
	Thinking About How Visual Studio .NET Uses a WSDL Document	119
	How Programmers Can Use the Proxy-Class Source Code	122
	Understanding the Simple Object Access Protocol (SOAP)	122
	Test Driving Web Services Methods within Your Browser	124
	Taking a Quick Look at Universal Description Discovery and Integration (UDDI)	127
	Summary	129
Chapter 5	Looking at Key Operations	131
	Using the Application Object	132
	Using the Application Object within a Web Service	135
	Using the Session Object	142
	Storing and Retrieving Cookie-Based Data	146
	Using Cookies within a Web Service	147
	Revisiting Session Objects	150
	Using Server Variables	152
	Removing a Web Service from an Application	157
	Summary	157
Part II	Publishing Your .NET Web Services	159
Chapter 6	Making .NET Web Services Available to Others	161
	Preparing to Publish Your Web Service	162
	Assigning a Unique Namespace to a Web Service	164
	Placing Web Services on the Web	165

	Understanding Universal Description Discovery Integration (UDDI)	166
	Taking a Closer Look at UDDI	169
	Taking Advantage of the UDDI Software Development Kit	170
	Helping Other Programmers Discover Your Web Services	172
	Understanding the .map File	174
	Summary	176
Chapter 7	Connecting Web Services to Databases	177
	Becoming Familiar with ADO.NET Operations	178
	Understanding the Data Provider	179
	Understanding the Steps to Access a Database	179
	Using a DataSet Object to Update a Database	185
	Using a Web Service to Interact with a Database	189
	Performing Database Operations from within an HTML Page	193
	Taking a Look at ADO.NET's Behind-the-Scenes Use of XML	195
	Building a DataSet Using XML-Based Data	197
	Retrieving a DataSet from an XML-Based File	203
	Summary	206
Chapter 8	Authenticating Users within .NET Services	207
	Allowing and Controlling Anonymous Access	208
	Understanding Basic Authentication	210
	Understanding Digest Authentication	213
	Understanding Windows Integrated Authentication	213
	Understanding Forms Authentication	213
	Understanding Certificate-Based Authentication	214
	Providing Username and Password Information	
	from within an Application	215
	Using the Web.config File to Configure Security Settings	218
	Summary	220
Chapter 9	Securing Communication between a Web Service and a Client	221
	Understanding the Threat	222
	Viewing HTTP Messages Firsthand	224
	Encrypting HTTP Messages Using the Secure Sockets Layer (SSL)	227
	Understanding Public-Key Encryption	228
	Obtaining Your Own Public and Private Key Pair	230

	Visiting a Secure Website	232
	Understanding Where SSL Resides within the Network Layers	233
	Looking Behind the Scenes at an SSL Connection	235
	Configuring a Web Server to Support Secure SSL-Based Connections	236
	Keep in Mind that SSL Protects the Exchange of Data Across the Network Only	237
	Connecting to a Web Service Using a Secure Connection	237
	Making an SSL Connection Mandatory for a Program to Use a Web Service	239
	Calling a Web Service Across an SSL Connection	243
	Summary	245
Chapter 10	Extending the Lines of Communication	247
	Sending an E-Mail Message with .NET	248
	Using a Web Service to Provide E-Mail Access	251
	Putting the MailAccess Web Service to Use	255
	Building an Online Chat Web Service	259
	Building a Chat Client	262
	Summary	264
Chapter 11	Integrating Binary Data into .NET Web Services	267
	Understanding the Process of Working with Binary Data	268
	Returning an Image File from a Web Service	270
	Requesting a File from the Web Service	272
	Displaying an Image File	274
	Providing Multimedia Files Using a Web Service	276
	Retrieving and Using Multimedia Data	277
	Receiving a Binary File Using a Web Service	281
	Uploading a Binary File to a Web Service	282
	Summary	284
Part III	Advanced Concepts	285
Chapter 12	Examining Key Web Service Files	287
	Taking a Quick Look at a Web Service's Support Files	288
	Using the AssemblyInfo File to Customize Assembly Settings	289
	Revisiting a Web Service's .asmx Files	294

	Using Global.asax to Specify Startup, Termination, and Request Processing	296
	Using the Web.config File to Customize Application Settings	297
	Using the .WebInfo File to Move a Web Service	300
	Using the .vsdisco File to Control a Web Service's Discovery	301
	Summary	302
Chapter 13	Unlocking Remote Access	303
	Providing Excel and Word Files Using a Web Service	304
	Retrieving an Excel or Word File	305
	Making Remote File Access More Flexible	308
	Installing the PlaceFileCopy Windows Service	314
	Using the PlaceFileCopy Windows Service	314
	Helping a Remote User Locate a File on the Server	319
	Searching a Server for a Specific File	320
	Locating and E-Mailing a File	322
	Sending a Remote File to an E-Mail Address	323
	Allowing a Web Service to Access Files on a User's PC	324
	Using a Web Service to Coordinate a Shared File's Use	325
	Summary	331
Chapter 14	Improving Web Page Performance	333
	Understanding the Big Picture (AKA How Programs Use Your Web Service)	334
	Using Events to Monitor Web Service Operations	335
	Monitoring a Web Service's Operations	337
	Using a Web Service to Monitor Specific Entries	339
	Using Caching to Improve Web Service Performance	342
	Understanding How Lack of a Cache Decreases a Web Service's Performance	344
	Caching a Web Service's Output	346
	Understanding When Caching a Web Service's Data Make Sense	346
	Understanding How Parameters Affect Web Service Caching	349
	Using Asynchronous Web Services	352
	Making Your First Asynchronous Call to a Web Service Method	353
	Calling the LocateFile Web Service Asynchronously	356
	Summary	359

Chapter 15	Making Money with Web Services	361
	Preparing the Database	362
	Offering and Managing User Keys	369
	Understanding the Overhead of Handling License Keys	371
	Taking a Close Look at the Registration Database	371
	Allowing Users to Upgrade a Web Service	375
	Handling Lost Keys	380
	Restricting a Web Service to 30 Days' Use	383
	Restricting a Web Service to Specific Hours of the Day	385
	Restricting the Number of Calls to a Web Service Each Day	388
	Billing on a Per-Use Basis	390
	Summary	393
Chapter 16	Putting It All Together	395
	Understanding How the Jobs Web Service Works	396
	Taking a Quick Look at the Employee Interface	396
	Taking a Quick Look at the Employer Interface	397
	Integrating a Database into the Jobs Web Service	399
	Controlling Site Access	399
	Building the Jobs Web Service	400
	Posting an RTF-Based Job or Resume	401
	Submitting RTF-Based Text to the Jobs Web Service	404
	Posting a Resume or Job Description Manually	407
	Submitting a Manual Resume or Job Description	410
	Deleting a Resume or Job Listing	412
	Retrieving Resumes and Job Opportunities	414
	Viewing Job Opportunities	418
	Viewing Employee Resumes	421
	Summary	425
	<i>Index</i>	427

Introduction:

Why Web Services Will Drive the Growth of the Web

Over the past 10 years, the World Wide Web has exploded into billions of pages of content. Each day, the Web touches our lives either directly, as we surf the Web, or indirectly, as companies use the Web to provide the products and services we consume.

Although HTML continues to evolve, many developers will agree that a key force that drove the Web's expansion was the ability for developers to automate web-page content using Active Server Pages. By automating content, Active Server Pages provided programmers with the ability to develop interactive e-commerce sites (such as Ebay and Amazon), information retrieval sites (such as Yahoo and Google), as well as sites that interacted with databases behind the scenes to place vast amounts of information within a user's browser.

Web services are the Web's next "big opportunity" for developers. You can think of a web service as a program that resides on a server that other programs can use to accomplish a specific result. The best way to understand how web services will change the way we use the Web is to envision having all the capabilities that users exploit on a site, such as the Yahoo! search engine, the Amazon shopping cart, or Southwest Airlines' ticketing capabilities, readily available within programs developers create. In other words, rather than having users leave your website to perform search operations at Yahoo, or to buy a book at Amazon or Barnes & Noble, your web pages (and the programs you create) can communicate with web services that reside on the remote sites so that your web pages (and programs) can offer the same functionality. In other words, by integrating support for a company's web services into your applications, your programs can provide users with search-engine capabilities, e-commerce support, and much more!

If you design your website correctly, users may have no need to leave your site. By taking advantage of web services, your web pages can offer the identical functionality the users would encounter at remote sites anywhere on the Web. It isn't difficult to imagine the immense power of web services, the promise they hold, or the great demands they will place on the skills and knowledge of Internet application developers.

Where .NET Comes into Play

This book examines web services within the .NET environment. You do not have to use .NET to create web services or to create programs that use web services. Many programmers, for example, use Java to create and call web services. However, the .NET environment makes the process of building and consuming web services so easy, I cannot imagine why web service programmers would want to use anything else.

If you have not yet installed Visual Studio .NET on your PC, that's okay. In Chapter 3, you will learn how to access web services from within your HTML pages using JavaScript!

What You Are Going to Learn

This book examines all aspects of web services. Web services are still a relatively new concept. As such, the book's early chapters focus on providing you with hands-on opportunities to interact with real-world web services that exist at sites across the Web. After that, you will learn how to create your own web services and how to make your web services available to other programmers across the Web.

No experience is required—with web services, that is. You'll learn from the ground up. If you are anxious to get started, go ahead and jump to Chapter 1 right now, and put a few web services you can find on the Web to work. This book, however, doesn't teach the languages used for creating web services. It assumes that you are an experienced VB.NET and C# programmer who wants to learn to put those skills to work developing web services.

Less than a decade ago, most of us would not have imagined that each day hundreds of millions of users would use their PCs to search worldwide for information, to purchase goods and services, or to work from their homes with access to the files and documents that reside on their office computers. Just as the Web has changed the way users interact with computers, web services provide a similar shift in how programmers might create applications in the future. For example, a programmer who needs the ability to translate text on the fly to a different language may integrate a translation web service into his or her program. Likewise, a programmer who must allow only authorized users to access a remote database that contains sensitive information might use a security web service to quickly implement the authentication.

Just as users search the Web to find information, programmers will use the web service discovery tools, such as the Universal Description, Discovery, and Integration (UDDI) protocol, to quickly locate the existing web services that offer a specific solution. This book will help you get ready for the new paradigm.

PART I

Laying the Foundation

Chapter 1: Taking Web Services for a Test Drive

Chapter 2: Creating Your First Web Services

Chapter 3: Accessing Web Services from within HTML Pages

Chapter 4: Looking Behind the Scenes at Web Service Protocols

Chapter 5: Looking at Key Operations



CHAPTER I

Taking Web Services for a Test Drive

- What's a Web Service?
- Understanding Operations That Are Well Suited for Web Services
- Retrieving Weather Information
- Using a Web Service 101
- Retrieving Stock Quotes
- Retrieving Book Information
- Retrieving Caller-ID Information
- Retrieving Traffic Information
- Retrieving Airport Information
- Where to Find Web Services on the Web

Unlike most discussions of web services that begin with an examination of the underlying network protocols, this chapter sets aside the underlying details and lets you test drive a variety of web services that other developers have created and made available on the Web. You will first experience many of the web services by using your browser to view active server pages that use the web service to implement their processing. Then, after you understand the operation the service performs, you will create a program that puts the service to use.

This chapter's goal is to give you a hands-on understanding of the types of web services you can create. This chapter makes extensive use of the Microsoft Visual Studio .NET programming environment to create programs that access the web services. You can take advantage of web services using several programming languages. This chapter presents programs and ASP.NET pages written in Visual Basic .NET and C#. If you are not yet programming within the .NET environment, the ease with which Visual Studio .NET lets you integrate web services into your programs should provide you with motivation to migrate to .NET.

In Chapter 2, "Creating Your First Web Services," you will learn that Visual Studio .NET also makes it easy for you to create your own web services. In Chapter 3, "Accessing Web Services from within HTML Pages," you will learn how to create HTML pages that interact with web services from within Visual Basic and applications that incorporate Visual Basic, such as Word and Excel. In later chapters, you will learn how to create and interact with web services using other programming languages such as Java and Perl.

What's a Web Service?

To break complex programs into manageable tasks, programmers make extensive use of *functions*. Each function within a program should perform specific processing (a service). For example, the following C program, `Hello.c`, uses the `printf` function to display a message to the user:

```
#include <stdio.h>

void main(void)
{
    printf("Hello, Programming World");
}
```

In a similar way, the following Visual Basic .NET code fragment, from the program `DisplayDateTime.vb`, uses the `Now` and `MessageBox.Show` functions to retrieve and then display the current date and time and the `Close` function to close the current form:

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As _  
    ➤ System.EventArgs) Handles MyBase.Load
```

```
MessageBox.Show("Current Date and time is: " & Now(), _  
    "Display Date & Time")
```

```
Me.Close()  
End Sub
```

Think of a web service as a function your programs can use to accomplish specific tasks. Just as a function can receive (and possibly change) parameter values, so too can a web service. Likewise, just as functions often return a value to the calling program, so too can a web service.

To use a function such as `printf` or `MessageBox.Show`, you must know the type of value the function returns as well as the number and types of parameters you can pass to the function. The same is true for a web service.

What makes a web service different from a traditional function is that the code for the web service resides on a remote server. Before a program can use a web service, the PC running the program must have a network connection (a dial-up connection will suffice).

When your program calls a web service (using a function call), your program, as shown in Figure 1.1, will send a network message to the server that specifies the desired service. If the web service requires parameters, the message will include values for each.

After the server completes the web service's processing, the server, as shown in Figure 1.2, will send a network message containing the service's result back to your program.

FIGURE 1.1:

To call a web service, a program sends a network message to the server upon which the web service resides.

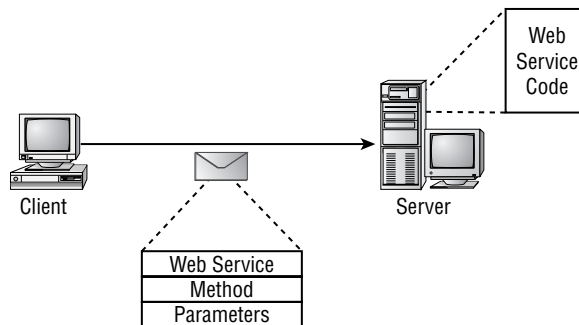
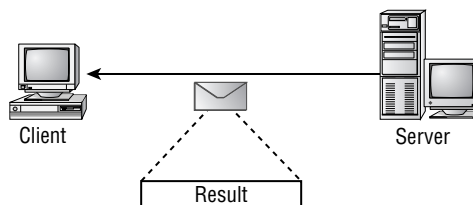


FIGURE 1.2:

After the server executes the web service's instructions, the server will send a network message containing the service's result to the calling program.



Because a web service requires the exchange of network messages and because the server that executes the service may be busy performing other tasks, a web service will execute substantially slower than a standard function. Depending on factors such as network traffic, the amount of time a web service will require may vary from one use of the service to the next.

Understanding Operations That Are Well Suited for Web Services

Because network overhead makes a web service execute much slower than a standard function, many operations are not well suited for implementation as a web service. Using a web service to add two numbers, for example, or to calculate a random number, would introduce unnecessary overhead to a program. Such operations are better suited for implementation using standard functions.

Web services, in contrast, are ideal for operations that use data residing at a remote source (most often within a database on a specific server). For example, web services are well suited for the following operations:

- Responding to queries for stock quotes
- Returning information regarding the availability of specific inventory items
- Providing real-time information, such as weather and road conditions
- Offering airline flight arrival and departure information
- Implementing e-commerce operations, such as ticket sales for movies and special events
- Authenticating users or credit-card information

You may be saying to yourself that users already perform such operations on many sites across the Web. Web services provide programmers with a way to integrate these operations into their programs. By using web services to implement common user operations (such as downloading stock quotes, ordering a book, and checking the weather) within your company's website, you can keep users from leaving your website to perform these operations elsewhere. By taking advantage of web services, you can integrate powerful processing developed by other programmers into your applications and web pages.

To help you better understand how web services extend the functionality of the Web into your applications, the remainder of this chapter will let you test drive readily available web services.

Web Service Updates from This Book's Companion Website

The examples this chapter presents make use of web services existing on the Web at the time of this writing. Over time, the services this chapter presents may change or become unavailable. This book's companion website, which you can find by following the links at www.sybex.com, will provide updates to the chapter code as the services change.

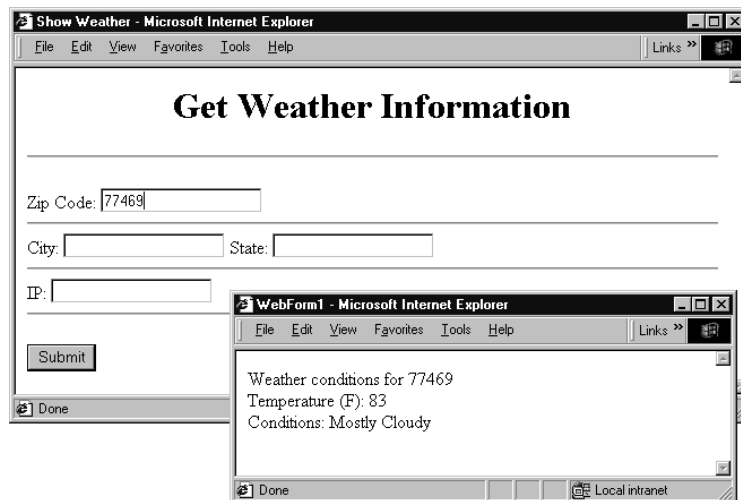
From the companion website, you can download the source code for all of the programs and services this book presents. You will find links to other key web development sites too.

Retrieving Weather Information

Each day millions of web users look up weather information. Across the Web, some of the fastest growing websites provide specifics about weather. The HTML file `ShowWeather.html`, which you can find at this book's companion website, creates a form that prompts the user to enter a zip code, city and state, or an Internet protocol (IP) address. After the user enters the data and clicks the Submit button, the user's browser sends the user input to an ASP.NET page that uses a Web service residing on the ServiceObjects website. The FastWeather web service will provide the ASP.NET page with weather data for a specific location. After the ASP.NET page receives the data from the service, it will display the result, as shown in Figure 1.3.

FIGURE 1.3:

Using a web service to obtain weather data



NOTE

The ServiceObjects website provides several powerful web services you can immediately integrate into your applications. Take time now to visit the site at www.ServiceObjects.com.

Looking Behind the Scenes at the FastWeather Web Service

To use the FastWeather web service, programs can call one of three methods (functions), passing to the methods the corresponding parameters:

```
string GetWeatherByIP(string IP, string LicenseKey)
string GetWeatherByCityState(string City, string State, string LicenseKey)
string GetWeatherByZip(string Zip, string LicenseKey)
```

Each of the functions, if successful (meaning the program provided a valid IP address, zip code, or city and state combination), will return a structure of type `Weather` that contains the following fields:

```
Weather
    string LastUpdated
    string TemperatureF
    string WindChill
    string HeatIndex
    string Humidity
    string Dewpoint
    string Wind
    string Pressure
    string Conditions
    string Visibility
    string Sunrise
    string Sunset
    string City
    string State
    string Moonrise
    string Moonset
    string Error
```

Note also that each of the FastWeather web service methods requires that you pass a parameter that specifies your license key. If you visit the ServiceObjects website, you can download a trial key that lets you use the service for a specific period of time. If you need unlimited use of the service, you must purchase a license for service from ServiceObjects. The `GetWeather.aspx` ASP.NET page uses the license key 0, which provides limited use of the service.

The source code in Listing 1.1 implements the ASP.NET page `GetWeather.aspx`.

**Listing 1.1****GetWeather.aspx**

```

Public Class WebForm1
    Inherits System.Web.UI.Page

    #Region " Web Form Designer Generated Code "
    ' Code not shown.
    #End Region

    Private Sub Page_Load(ByVal sender As System.Object, _
        ➤ ByVal e As System.EventArgs) Handles MyBase.Load
        Dim Zip, City, State, IP As String
        Dim WebError As Boolean = False
        Dim QueryPerformed As Boolean = True

        Zip = Request.Form("ZipCode")
        City = Request.Form("City")
        State = Request.Form("State")
        IP = Request.Form("IP")

        Dim WeatherRequest As New net.serviceobjects.ws.FastWeather()
        Dim Weather As net.serviceobjects.ws.Weather

        Try
            If (Zip <> "") Then
                Response.Write("Weather conditions for " & Zip)
                Weather = WeatherRequest.GetWeatherByZip(Zip, 0)
            ElseIf (City <> "") And (State <> "") Then
                Response.Write("Weather conditions for " & City & _
                ➤ " " & State)
                ➤ Weather = WeatherRequest.GetWeatherByCityState(City, _
                    State, 0)
            ElseIf (IP <> "") Then
                Response.Write("Weather conditions for " & IP)
                Weather = WeatherRequest.GetWeatherByIP(IP, 0)
            Else
                Response.Write("Must specify valid location")
                QueryPerformed = False
            End If

            Catch Ex As Exception
                Response.Write("Web service error: " & Ex.Message)
                WebError = True
            End Try

            If (Not WebError And QueryPerformed) Then
                If (Weather.Error = "") Then
                    Response.Write("<br/>")
                    Response.Write("Temperature (F): " & _
                    Weather.TemperatureF & "<br/>")
                End If
            End If
        End Sub
    End Class

```

```
        Response.Write("Conditions: " & Weather.Conditions)
    Else
        Response.Write("<br/>")
        Response.Write("Web service returned an error: " & _
            Weather.Error)
    End If
End If

End Sub

End Class
```

As you can see, the code first uses the *Request* object to determine the values the user assigned to the zip code, city, state, or IP fields. To use a web service, a program must create an object specific to the service. The following statement creates a variable named *WeatherRequest* that corresponds to the FastWeather service:

```
Dim WeatherRequest As New net.serviceobjects.ws.FastWeather()
```

Throughout this chapter, you will create similar objects for the different web services. The object name, in this case *net.serviceobjects.ws.FastWeather*, identifies the web service. As you will see when you create a C# program that uses the FastWeather web service, Visual Studio .NET makes it easy for you to determine the object name.

As discussed, the FastWeather web service returns a value of type *Weather* that contains the individual weather fields. The following statement defines a variable to store the Weather structure:

```
Dim Weather As net.serviceobjects.ws.Weather
```

To use a web service, you simply call one of the methods the service provides. In this case, the code uses an If-Else statement to determine which method to call based on whether the user specified a zip code, city and state, or IP address. The following statement, for example, calls the service's *GetWeatherByZipCode* method:

```
Weather = WeatherRequest.GetWeatherByZip(Zip, 0)
```

Note that the code calls the web service methods within a Try-Catch block. Most web services will generate an exception when an error occurs. When your programs call a web service, they should always do so within a Try-Catch block so your code can detect and respond to an exception generated by the service.

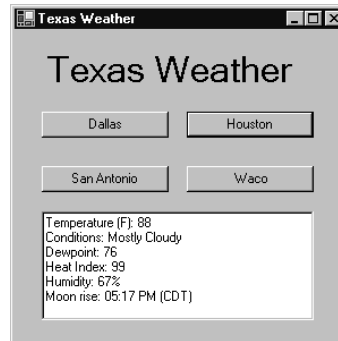
The application uses an ASP.NET page, as opposed to an active server page, because of the ease with which Visual Studio .NET lets developers integrate a web service.

Retrieving Weather Information within a C# Program

Web services exist to help programmers integrate web-based operations into their programs. The Visual Basic .NET program, `TexasWeather.vb`, displays a form that contains buttons corresponding to Texas cities. After the user clicks a button, the program displays the corresponding weather data, as shown in Figure 1.4.

FIGURE 1.4:

Using the FastWeather web service within a Visual Basic .NET program



In this case, the program uses only the FastWeather service's `GetWeatherByCityState` method. To create the `TexasWeather.vb` program, perform these steps:

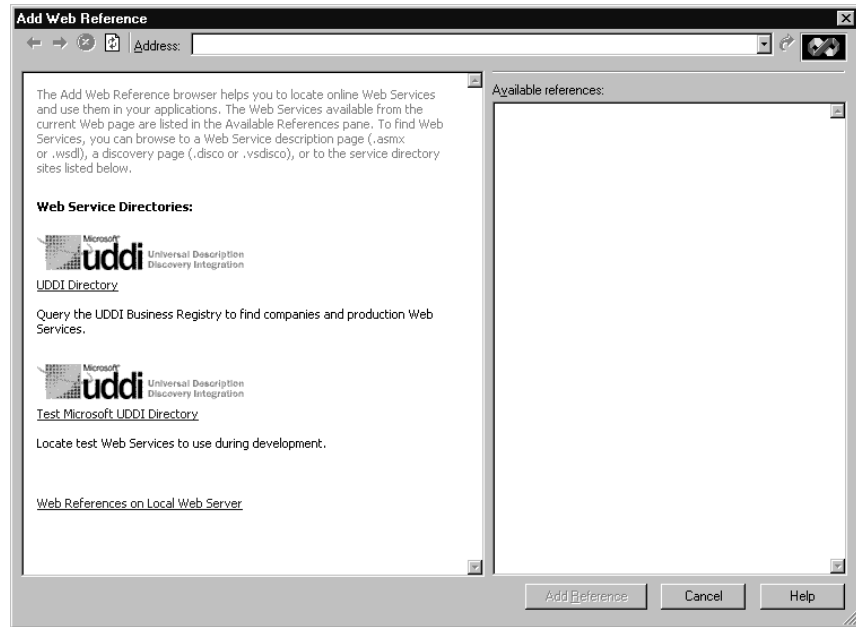
1. Within Visual Studio .NET, select the File menu New Project option. Visual Studio .NET will display the New Project dialog box.
2. Within the New Project dialog box Project Types list, click Visual C# Projects. Then, within the Templates field, click Windows Application. Finally, within the Location field, specify the folder within which you want to store the program and the program name **TexasWeather**. Select OK. Visual Studio .NET will display a form onto which you can drag and drop the program's controls (label, buttons, and text box).
3. Using the Toolbox, drag and drop the label, buttons, and text box previously shown in Figure 1.4 onto the form.
4. To use a web service, you must assign a Web Reference to the program that corresponds to the object. To do so, select the Project menu Add Web Reference option. Visual Studio .NET will display the Add Web Reference dialog box, as shown in Figure 1.5.

NOTE

Over time, the URLs this book uses (such as the one in Step 5) for the WSDL (web service definition language) files that describe a web service may change. See the section "Using a Web Service 101" to determine the URL you should enter for a service's WSDL file within the Add Web Reference dialog box.

FIGURE 1.5:

The Add Web Reference dialog box



5. Within the Address field, you must type the URL of a special file (called the WSDL file) that describes the web service. In this case, type **`http://ws.serviceobjects.net/fw/FastWeather.asmx?WSDL`** and press Enter. The dialog box will load the file's contents. Click the Add Reference button.
6. Select the View menu Code option. Visual Studio .NET will display the program's source code. Within the source code (near the bottom of the GetWeather class definition), add the following program statements:

```
private void GetWeather(String City, String State)
{
    Boolean WebError = false;

    net.serviceobjects.ws.FastWeather WeatherRequest;
    net.serviceobjects.ws.Weather Weather = null;
    WeatherRequest = new net.serviceobjects.ws.FastWeather();

    try
    {
        Weather = WeatherRequest.GetWeatherByCityState(City, _
➡          State, "0");
    }
    catch (Exception Ex)
```

```

    {
        textBox1.Text = "Web service error: " + Ex.Message;
        WebError = true;
    }

    if (! WebError)
    {
        if (Weather.Error == null)
        {
            textBox1.Text = "Location: " + Weather.City + "\r\n";
            textBox1.Text = "Temperature (F): " + _
➡       Weather.TemperatureF + "\r\n";
            textBox1.Text += "Conditions: " + _
➡       Weather.Conditions + "\r\n";
            textBox1.Text += "Dewpoint: " + Weather.Dewpoint + "\r\n";
            textBox1.Text += "Heat Index: " + _
➡       Weather.HeatIndex + "\r\n";
            textBox1.Text += "Humidity: " + Weather.Humidity + "\r\n";
            textBox1.Text += "Moon rise: " + Weather.Moonrise + _
➡       "\r\n";
            textBox1.Text += "Moon set: " + Weather.Moonset + "\r\n";
            textBox1.Text += "Pressure: " + Weather.Pressure + "\r\n";
            textBox1.Text += "Sun rise: " + Weather.Sunrise + "\r\n";
            textBox1.Text += "Sun set: " + Weather.Sunset + "\r\n";
            textBox1.Text += "Visibility: " + Weather.Visibility + _
➡       "\r\n";
            textBox1.Text += "Wind: " + Weather.Wind + "\r\n";
            textBox1.Text += "Wind chill: " + Weather.Windchill;
        }
        else
        {
            textBox1.Text = "Web service returned an error: " + _
➡       Weather.Error;
        }
    }

    private void Form1_Load(object sender, System.EventArgs e)
    {

    }

    private void button1_Click(object sender, System.EventArgs e)
    {
        GetWeather("Dallas", "TX");
    }

    private void button2_Click(object sender, System.EventArgs e)
    {

```

```
        GetWeather("Houston", "TX");
    }

    private void button3_Click(object sender, System.EventArgs e)
    {
        GetWeather("San Antonio", "TX");
    }

    private void button4_Click(object sender, System.EventArgs e)
    {
        GetWeather("Waco", "TX");
    }
}
```

The program provides an event handler that responds to each user button click. Within each handler, the code calls the `GetWeather` function, passing to the function the name of a specific city and the *TX* state abbreviation. Within the `GetWeather` function, the following statements create an object named `WeatherRequest` that the program will use to access the `FastWeather` web service:

```
net.serviceobjects.ws.FastWeather WeatherRequest;
WeatherRequest = new net.serviceobjects.ws.FastWeather();
```

Again, the `FastWeather` web service returns a value of type `Weather`. The following statement creates a variable that will hold the specific weather fields:

```
net.serviceobjects.ws.Weather Weather = null;
```

The program calls the web service method `GetWeatherByCityState` within a try-catch block to detect any exceptions the web service may generate:

```
try
{
    Weather = WeatherRequest.GetWeatherByCityState(City, _
    ➡ State, "0");
}
catch (Exception Ex)
{
    textBox1.Text = "Web service error: " + Ex.Message;
    WebError = true;
}
```

Finally, if the web service is successful, the code displays the various weather elements within a text box.

Using a Web Service 101

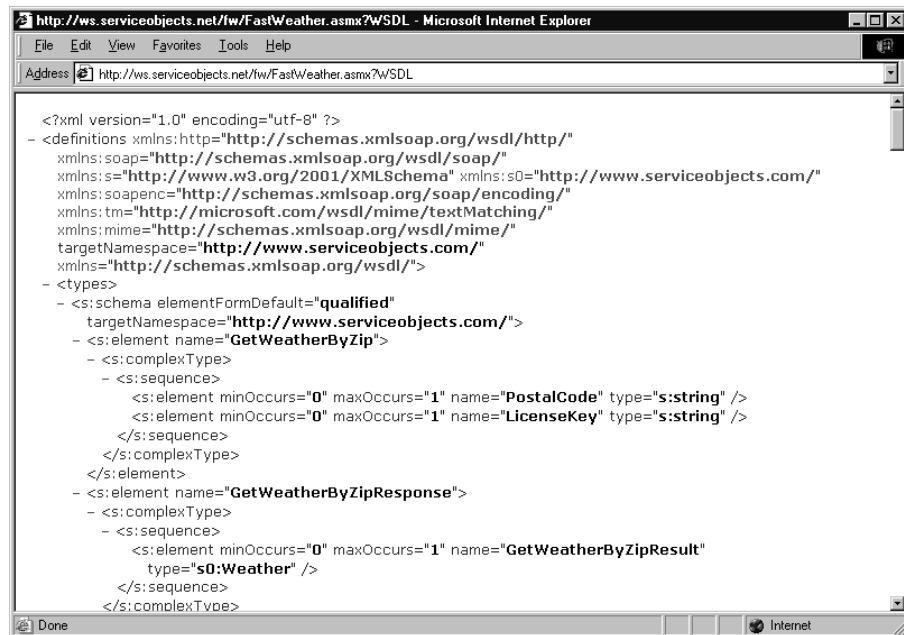
Using a web service within a .NET program is a straightforward process. To begin, you must add to the program a Web Reference that corresponds to the web service. To do so, you must know the location of the WSDL file that describes the web service.

If you examine websites that make web services available to programmers, you will find that the sites always contain a link to the web service's WSDL file. If you click the link, your browser will display XML-based data that describe the service, similar to that shown in Figure 1.6. XML, as you know, is the Extensible Markup Language that developers use to describe data. Throughout this book, you will make extensive use of WSDL, the web service definition language. For now, think of WSDL as providing a description of the methods (functions) a web service provides, as well as a description of the parameters each method requires.

For now, you can ignore the XML statements that describe the service. Instead, note the Web address that appears within the browser's address field. You can either write down the address or cut and paste the address into the Visual Studio .NET Add Web Reference dialog box.

FIGURE 1.6:

Viewing a web service's WSDL file



After you add a Web Reference to your program code for the web service, you must then create a corresponding object within your source code. The following statement creates an object that a Visual Basic .NET program can use to interact with the FastWeather web service:

```
Dim WeatherRequest As New net.serviceobjects.ws.FastWeather()
```

After you add a service's Web Reference to your program, Visual Studio .NET will display the reference within the Class View window. As you view the web services within the Class View window, you will find that most web service object names will begin with letter combinations such as *net*. If you simply type the first two letters of the name, Visual Studio .NET usually will display the service's remaining characters, making it very easy for you to enter the correct object names.

Regardless of the web service you want to use from within a .NET program and regardless of whether you are writing the program using Visual Basic .NET and C# or if you are creating an ASP.NET page, the steps you will perform are the same. If your code requires multiple web services, you must perform these steps for each object.

Retrieving Stock Quotes

Across the Web, many websites offer users the ability to retrieve stock information for a specific company. To comply with securities regulations, the stock information is delayed by 15 minutes.

The StockQuote web service, available from the XMethods website at www.xmethods.com, retrieves delayed stock prices for the company that corresponds to a stock symbol, such as *MSFT* for Microsoft.

NOTE

The XMethods website at www.xmethods.com provides many web services you can integrate into your applications. Take time to visit the XMethods website—you will likely find several web services you can put to immediate use.

The ASP.NET page *StockPrice.aspx*, which you can run from this book's companion website, displays the form shown in Figure 1.7 that contains buttons corresponding to several software companies. When the user clicks a button, the page will display the company's (delayed) stock price.

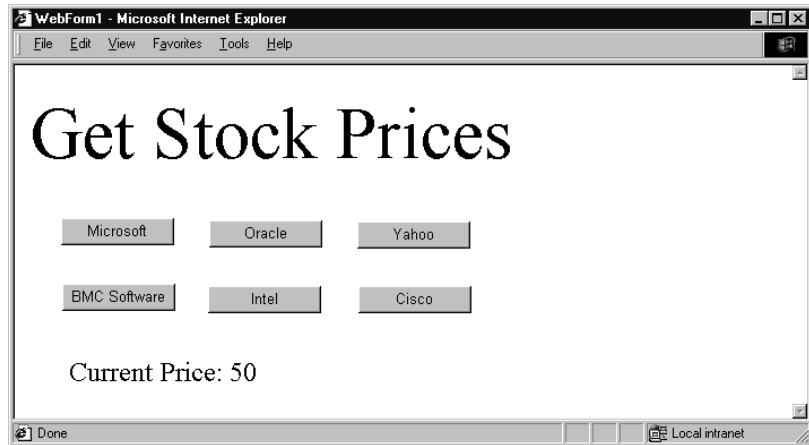
Looking Behind the Scenes at the StockQuote Web Service

The StockQuote web service supports one method, *getQuote*, which returns a value of type *Float* that corresponds to a company's stock price. Your programs pass the stock symbol, such as *MSFT* for Microsoft, to the method as a parameter:

```
float getQuote(string symbol)
```

FIGURE 1.7:

Using the StockQuote web service to retrieve stock prices



If the symbol your program passes to `getQuote` is invalid, `getQuote` will return the value `-1`. The source code in Listing 1.2 implements the ASP.NET page `StockPrice.aspx`.

**Listing 1.2****StockPrice.aspx**

```
Public Class WebForm1
    Inherits System.Web.UI.Page
    Protected WithEvents Label1 As System.Web.UI.WebControls.Label
    Protected WithEvents Button1 As System.Web.UI.WebControls.Button
    Protected WithEvents Button2 As System.Web.UI.WebControls.Button
    Protected WithEvents Button3 As System.Web.UI.WebControls.Button
    Protected WithEvents Button4 As System.Web.UI.WebControls.Button
    Protected WithEvents Button5 As System.Web.UI.WebControls.Button
    Protected WithEvents Label2 As System.Web.UI.WebControls.Label
    Protected WithEvents Button6 As System.Web.UI.WebControls.Button

    #Region " Web Form Designer Generated Code "
        'Code not show
    #End Region

    Private Sub Button1_Click(ByVal sender As System.Object, _
        ➔ ByVal e As System.EventArgs) Handles Button1.Click
        ShowStockPrice("MSFT")
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, _
        ➔ ByVal e As System.EventArgs) Handles Button2.Click
        ShowStockPrice("ORCL")
    End Sub
```

```

    Private Sub Button3_Click(ByVal sender As System.Object, _
➡   ByVal e As System.EventArgs) Handles Button3.Click
        ShowStockPrice("YH00")
    End Sub

    Private Sub Button4_Click(ByVal sender As System.Object, _
➡   ByVal e As System.EventArgs) Handles Button4.Click
        ShowStockPrice("BMC")
    End Sub

    Private Sub Button5_Click(ByVal sender As System.Object, _
➡   ByVal e As System.EventArgs) Handles Button5.Click
        ShowStockPrice("INTC")
    End Sub

    Private Sub Button6_Click(ByVal sender As System.Object, _
➡   ByVal e As System.EventArgs) Handles Button6.Click
        ShowStockPrice("CSCO")
    End Sub

    Private Function ShowStockPrice(ByVal Symbol As String) _
➡   As String
        Dim StockQuote As New _
➡   net.xmlmethods.services.netxmlmethodsservicesstockquoteStockQuoteService()
        Dim Price As String

        Price = StockQuote.getQuote(Symbol)

        Label2.Text = "Current Price: " & Price
    End Function

End Class

```

As you can see, the code provides event handlers for each of the buttons. Within the handler, the code calls the `ShowStockPrice` function, passing to the function a stock symbol that corresponds to a specific company. Within the `ShowStockPrice` function, the following statement creates a variable named *StockQuote* that corresponds to the object the code will use to interact with the *StockQuote* object:

```

Dim StockQuote As New _
➡   net.xmlmethods.services.netxmlmethodsservices_
➡   stockquoteStockQuoteService()

```

To call the `getQuote` method, the code uses the *StockQuote* variable as follows:

```

Price = StockQuote.getQuote(Symbol)

```


Retrieving Stock Prices within a C# Program

The following C# program, `GetQuote.cs`, displays a form that prompts the user for a company stock symbol. After the user enters the symbol and clicks the Get Stock Price button, the program will display the stock price, as shown in Figure 1.8.

FIGURE 1.8:

Using the StockQuote web service within a C# program



To create the `GetQuote.cs` program, perform these steps:

1. Within Visual Studio .NET, select the File menu New Project option. Visual Studio .NET will display the New Project dialog box.
2. Within the New Project dialog box Project Types list, click Visual C# Projects. Then, within the Templates field, click Windows Application. Finally, within the Location field, specify the folder within which you want to store the program and the program name **GetQuote**. Select OK. Visual Studio .NET will display a form onto which you can drag and drop the program's controls (label, buttons, and text box).
3. Using the Toolbox, drag and drop the label, buttons, and text box previously shown in Figure 1.8 onto the form.
4. To assign a Web Reference that corresponds to the object, select the Project menu Add Web Reference option. Visual Studio .NET will display the Add Web Reference dialog box.
5. Within the Address field, type the URL of the service's WSDL file. In this case, type **`http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl`** and press Enter. The dialog box will load the file's contents. Click the Add Reference button.
6. Select the View menu Code option. Visual Studio .NET will display the program's source code. Within the source code (near the bottom of the class definition), add the following program statements:

```
private void button1_Click(object sender, System.EventArgs e)
{
    float Price;
    net.xmethods.services.netxmethodsservicesstockquote
➡    StockQuoteService Quote;
```

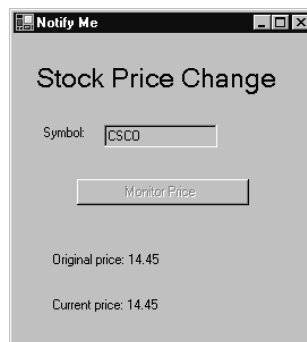
```
        Quote = new
➡    net.xmlmethods.services.netxmlmethodsservicesstockquoteStockQuoteService();

    if (textBox1.Text.Length == 0)
        label2.Text = "Must specify stock symbol";
    else
    {
        try
        {
            Price = Quote.getQuote(textBox1.Text);
            if (Price == -1)
                label2.Text = "Invalid symbol";
            else
                label2.Text = "Current price: " + Price.ToString();
        }
        catch(Exception ex)
        {
            label2.Text = "Web service exception" + ex.Message;
        }
    }
}
```

The previous program used the StockQuote web service to retrieve a stock price for display. The following C# program, NotifyMe.cs, again prompts the user to enter a stock symbol. After the user enters the stock information, the user can minimize the program. The program, behind the scenes, will “wake up” every 15 seconds and compare the stock’s current price to the original price. If the stock’s price has changed by one dollar (either up or down), the program will bring the form to the top of any active programs. If the user has minimized the program, the program will highlight the program’s icon within the Taskbar. See Figure 1.9.

FIGURE 1.9:

Using a web service within a program that performs background processing



To create the NotifyMe.cs program, perform these steps:

1. Within Visual Studio .NET, select the File menu New Project option. Visual Studio .NET will display the New Project dialog box.
2. Within the New Project dialog box Project Types list, click Visual C# Projects. Then, within the Templates field, click Windows Application. Finally, within the Location field, specify the folder within which you want to store the program and the program name **NotifyMe**. Select OK. Visual Studio .NET will display a form onto which you can drag and drop the program's controls (label, buttons, and text box).
3. Using the Toolbox, drag and drop the label, buttons, and text box previously shown in Figure 1.9 onto the form. Then, drag a Timer control onto the form.
4. Select the Project menu Add Web Reference option. Visual Studio .NET will display the Add Web Reference dialog box.
5. Within the Address field, type the URL **<http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl>** and press Enter. The dialog box will load the file's contents. Click the Add Reference button.
6. Select the View menu Code option. Visual Studio .NET will display the program's source code. Within the source code (near the bottom of the class definition), add the following program statements:

```
float OriginalPrice;
float DollarChange;

net.xmethods.services.netxmethodsservicesstockquote
➡ StockQuoteService Quote;

private void button1_Click(object sender, System.EventArgs e)
{
    float Price;
    Quote = new
➡ net.xmethods.services.netxmethodsservicesstockquoteStockQuoteService();

    if (textBox1.Text.Length == 0)
        label4.Text = "Must specify stock symbol";
    else
    {
        textBox1.ReadOnly = true;
        button1.Enabled = false;

        try
        {
            Price = Quote.getQuote(textBox1.Text);
            OriginalPrice = Price;
```

```
        if (Price == -1)
            label4.Text = "Invalid symbol";
        else
        {
            label4.Text = "Original price: " + OriginalPrice.ToString();
            timer1.Interval = 15000;
            timer1.Enabled = true;
            DollarChange = 0;
        }
    }
    catch(Exception ex)
    {
        label4.Text = "Web service exception" + ex.Message;
    }
}

private void timer1_Tick(object sender, System.EventArgs e)
{
    float Price;

    try
    {
        Price = Quote.getQuote(textBox1.Text);

        if (Price == -1)
        {
            label5.Text = "Invalid symbol";
            this.Activate();
        }
        else
        {
            label5.Text = "Current price: " + Price.ToString();

            if (((Price - OriginalPrice) > DollarChange) ||
                ((OriginalPrice - Price) > DollarChange))
            {
                this.Activate();
                this.BringToFront();
            }
        }
    }
    catch(Exception ex)
    {
        label5.Text = "Web service exception" + ex.Message;
    }
}
```

The program uses a timer set to 15-second intervals. Each time the timer occurs, the code uses the web service to retrieve the stock's current price. If the stock price has increased or decreased by a dollar or more since the user first requested the price, the code will bring the form to the top of any open applications:

```
if (((Price - OriginalPrice) > DollarChange) ||  
    ((OriginalPrice - Price) > DollarChange))  
{  
    this.Activate();  
    this.BringToFront();  
}
```

Retrieving Book Information

On the Web, Amazon (amazon.com) and Barnes & Noble (barnesandnoble.com) are two of the largest online booksellers. Both sites let users shop for books electronically. To integrate the capabilities of these two online sites into your own programs and web pages, you can take advantage of web services.

To start, Amazon offers a software development kit (SDK) programmers can use to search for books, videos, and music, and by keyword, author, artist, and more. Further, programmers can integrate support for the Amazon shopping cart into their own applications and websites.

You can download the Amazon web services SDK from the Amazon website at www.amazon.com/webservices. After you download the software development kit, you must apply for a developer's token (a key) that you must include as a parameter within your function calls to the services.

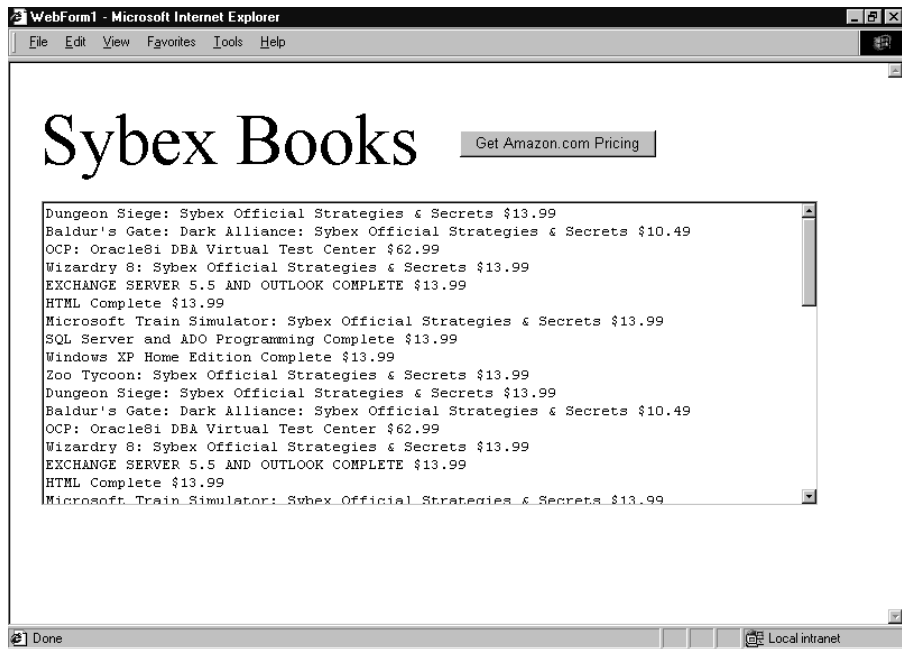
Second, the BNQuote web service returns the price of a book at Barnes & Noble for a given ISBN.

The ASP.NET page `AmazonDemo.aspx` on this book's companion website uses the Amazon web services to list the titles and prices of various Sybex books at Amazon. When you display the page and click on the Get Amazon.com Pricing button, the page will use the Amazon web service to perform a keyword search on "Sybex." The page will place the search results for the first 50 books within the text box, as shown in Figure 1.10.

Likewise, the ASP.NET page `BarnesAndNoble.aspx` at this book's companion website displays buttons for several different book titles. If you click one of the buttons, the page will display the book's current price at the online Barnes & Noble store, as shown in Figure 1.11.

FIGURE 1.10:

Using the Amazon web services SDK within an ASP.NET page

**FIGURE 1.11:**

Using the BNQuote Web service to display book prices at Barnes & Noble online

