CHAPTER 1

Introducing XML Web Services and the .NET Framework

IN THIS CHAPTER:
Beyond the Web Services Hype
Web Services Architectures
Core XML Web Service “Standards”
XML Web Services Support in .NET
A week seldom goes by without a deluge of press releases, online and print articles, and white papers covering the “Web services revolution.” Information technology pundits and computer journalists breathlessly proclaim Web services to be the “next big thing” in computing technology. For example, Gartner Group, a leading technology analysis and advisory firm, rates Web services as one of “... four key emerging technology trends for the next decade.” Following is an excerpt from Gartner’s October, 2001, press release:

By packaging business processes as software components, Web services will drive much of the still-to-be-developed e-business landscape. Web services will facilitate much faster software development and integration. They will also enable businesses to become more agile, and help them focus on their core competencies while outsourcing other functions. Web services are likely the hottest trend of 2001 and 2002, and are probably still an underestimated technology.

NOTE
For a comprehensive list of Gartner’s Web service reports and news articles, go to http://www3.gartner.com/ and search for “Web Services.”

“[P]ackaging business processes as software components” is old hat. Visual Basic developers have been packaging business processes as business logic components since 1995, when Microsoft added the ability to create COM DLLs to Visual Basic 4.0. Microsoft Transaction Server enabled multiple COM components to enlist in distributed database transactions, which made three-tier application architecture a reality and enabled Visual Basic developers to write enterprise-class applications. Active Server Pages and server-side VBScript or ECMAScript deliver the presentation layer to browser-based, “thin” clients that connect to middle-tier business-process components. Any Internet-based application that takes advantage of object-oriented architecture and uses HTTP or other standard Internet protocols to communicate with other servers or deliver the presentation layer to clients qualifies as a generic Web service. Web-based collaboration schemes and peer-to-peer sharing of idle computer resources can also qualify as Web-based services. Thus, generic Web services aren’t a new technology.

A critical prefix is missing from most of today’s Web services prose. The technology that writers and analysts are gushing over is XML Web services. Industry-standard Extensible Markup Language (XML) documents, the Simple Object Access Protocol (SOAP), and Web Services Description Language (WSDL) files distinguish XML Web services from their operating system–specific, protocol-constrained, and language-dependent predecessors. This book is about XML Web services running under Windows 2000+ and is intended for Visual Basic 6.0 and current or prospective Visual Basic .NET and ASP.NET developers. The XML Web services you create and test as you progress through this book are accessible to any client or server that can communicate via HTTP, HTTPS, or SMTP and process the XML content of a WSDL file.
Beyond the Web Services Hype

Most press and analyst reports, including .NET propaganda from Microsoft, have focused on electronic commerce (e-commerce) as the driving force behind the development of [XML] Web services. It’s impossible to accurately determine how many billions of dollars now-bankrupt dot-coms have thrown into business-to-consumer (B2C) e-commerce’s bottomless money pit. To date, business-to-business (B2B) e-commerce participants, such as commodity- and industry-specific exchanges (and their software suppliers) haven’t fared much better than their B2C predecessors. Today’s most serious impediment to adoption of XML Web services technology by IT management is the early e-commerce stigma.

Enterprise (application) integration is the key to initial deployment of XML Web services. Enterprise integration currently has much broader scope than e-commerce applications. The goal of enterprise integration is interoperability of distributed computing systems of all types and sizes, not just B2C or B2B projects. The overwhelming majority of distributed systems involve interaction between databases that reside on mainframes, UNIX boxes, and Windows NT/2000 servers. XML Web services map the route to the Holy Grail of Database Interoperability and Software Reusability, and promise a substantial increase in the return on investment (ROI) of software development projects.

ARC Advisory Services, a strategic planning and technology assessment firm, pegs the 2006 enterprise integration software and services market at $3.9 billion in 2000, $4.8 billion in 2001, and over $11 billion by 2006. The increase in the enterprise integration market stands in sharp contrast to the cutbacks in overall information technology spending in late 2001 and early 2002. ARC’s October, 2001, press release states:

Web [s]ervices are on the horizon with strong support from the J2EE-based suppliers as well as Microsoft with its .NET platform. Web services will provide a standard communications interface between heterogeneous platforms, and is well suited for Internet-based applications and [e]nterprise [i]ntegration.

In March 2002, Gartner Group projected that the overall market for Web services will reach $28 billion by 2006. If both the ARC and Gartner estimates are correct, B2B and B2C Web services will garner more than $17 billion in 2006. Market size predictions for new technologies are notoriously inaccurate, but B2C and, to a lesser extent, B2C services will probably gain the largest share of IT spending for Web services by 2004.

TIP

If you (or, more importantly, your bosses) believe an article printed in the Harvard Business Review is to business management as a paper published in the New England Journal of Medicine is to health care, spend $6.00 to download the PDF version of the October 2001 “Your Next IT Strategy” article by John Hagel III and John Seelye Brown from http://www.hbsp.harvard.edu/hbsp/prod_detail.asp?R0109G. The article, which is directed at corporate and IT management, not developers, describes an analogy between “service grids” of XML Web services and electrical power distribution grids. Put all members of your organization’s upper management on the distribution list for this article.
Web Services Architectures

Web services permit programming objects to communicate across a public network (the Internet) where the endpoints (hosts) are located behind firewalls. Virtually all firewalls permit HTTP request-response response operations by clients on TCP port 80, and many support encrypted communication by the Secure Sockets Layer (SSL) API and secure HTTP (HTTPS) protocol on port 443. HTTP is a stateless (also called connectionless) protocol, which contributes to the scalability of Web-based applications; thus, all Web services are said to be “loosely coupled.” In this respect, Web services are related to messaging services, but not to message queuing or store-and-forward applications. If your Web-based applications require maintaining session-level or application-level state, the Web server, not the Web service, stores state information for the client.

NOTE

The term “Web service” implies the “World Wide” prefix and conformance with applicable World Wide Web Consortium (W3C) recommendations, such as those for HTML and XHTML. Services using Internet transport protocols other than HTTP, TCP, and UDP or remote procedure call (RPC) methods are beyond the scope of Web services. Specifically, FTP and SMTP transports aren’t covered in this book.

Another characteristic of all Web services is that, unlike most ActiveX controls and Java applets, Web services don’t interact directly with the user interface, and many Web services are intended only for server-to-server communication. Thin (browser-based) and thick (Windows application-based) clients provide the UI. You compile Visual Basic 6.0 ActiveX DLLs that participate in Web services with the Unattended Execution and Retained in Memory options. The Unattended Execution option prohibits inclusion of forms in your DLL, and use of message and input boxes in your code. Similar restrictions apply to XML Web services you create with ASP.NET and Visual Basic .NET.

The following sections describe by example the differences between generic and XML Web services.

Generic Web Services

Internet Information Server’s SQLXML 3.0 add-on provides the capability to create data-related generic Web services. A COM or .NET DLL can use connectionless HTTP or HTTPS wire protocols to POST the text of a conventional SQL query with a FOR XML AUTO modifier appended. The FOR XML AUTO modifier returns the query result set as an XML document. Template queries enhance the data-retrieval process by defining a Transact-SQL query that accepts WHERE clause constraints expressed as arguments of HTTP requests. SQL Server 2000 XML updategrams handle connectionless updates to remote databases and support transactions.
Chapter 1: Introducing XML Web Services and the .NET Framework

NOTE

Formatting a query result set or updategram as an XML document doesn’t qualify either of the two processes as an XML Web service, as defined in the following section. Microsoft could have chosen any standard text format, such as comma-separated values (CSV) or Rich Text Format (RTF) files, to enable data transport via HTTP. Remote Data Services (RDS), which Microsoft now terms a “deprecated” (read “no longer supported”) technology, uses MIME encoding to pass lightweight ADO.Recordset objects through firewalls.

Figure 1-1 illustrates three approaches to retrieving data from and updating remote SQL Server databases protected by firewalls:

► A client’s browser issues a GET or POST request to an .asp or .aspx (ASP.NET) page over an intranet. The page’s VBScript or Visual Basic .NET code sends an HTTP GET request to the virtual directory to specify the SQL Server template and the parameter value(s). SQL Server returns the query result set as attribute-centric XML, which the component transforms to an HTML or, preferably, XHTML table. A Response.Write str[X]HTML instruction sends the table to the client’s browser.

► A client’s browser opens an HTML page that uses dynamic HTML to generate a template query based on user input. The browser sends the HTTP request directly to a different template, which specifies an XSL file to transform the XML to an HTML table on the remote server. The server responds to the client directly.

► A Visual Basic application running on a Windows client connects to a remote ActiveX DLL via DCOM. The component translates client requests for data into template queries and updates into updategrams. Template queries return XML data to the component for translation into a format that’s compatible with the client application. For example, the component can append the XML query result set to a set of custom header and schema elements and deliver a persistable XML ADODB.Recordset object to the client.

TIP

For more details on SQLXML template queries and transforms, go to http://www.oakleaf.ws/articles and click the Enterprise 2001 link.

The first two of the preceding data access scenarios are operating system–independent. For example, an Opera browser running under Linux or Internet Explorer (IE) on a Macintosh delivers the same result as IE on Windows desktops. The second data access method doesn’t depend on ASP or ASP.NET, so this approach is totally operating system–agnostic. The last example is Windows-centric and the most difficult to deploy. Configuring DCOM proxies on thousands of Windows 95, 98, Me, NT, and 2000 clients is a very difficult and costly project. Dispensing with DCOM is one of the primary incentives for adopting XML Web services as your standard component access technology.
Figure 1-1  IIS’ XML for SQL Server add-on enables Web servers and browser-based or Windows clients to access remote, firewall-protected SQL Server databases by HTTP GET and POST operations.
XML Web Services

“XML Web services” is Microsoft’s official term for the subject of this book, although you’re likely to see “Web services,” “.NET Web services,” and “ASP.NET Web Services” in early Microsoft .NET Framework documentation and developer presentations. XML Web services require use of SOAP for communication, WSDL files for definition of functionality, XML for content (called payload), and XML Schema Definition (XSD) language for datatype definitions. ASP.NET Web Service projects, which generate instances of System.Web.Services objects, conform to these requirements, so you can consider .NET and ASP.NET a synonym for XML in this context. Plain “Web services” appears mostly in early Microsoft publications that haven’t been reedited to conform to today’s terminology.

NOTE
When you create a new ASP.NET Web Service project, Visual Studio .NET generates its WSDL file automatically. Early beta versions of Visual Studio .NET employed Microsoft’s proprietary Service Description Language (SDL). Use of the term “ASP.NET Web Service” in this book refers only to ASP.NET projects for creating XML Web services that use SOAP messaging, with a few explicit exceptions. The “System.Web.Services Namespaces” and “ASP.NET Web Service Projects” sections near the end of this chapter offer more detailed coverage of this issue.

Figure 1-2 illustrates the following data-related XML Web service scenarios that correspond approximately to those shown in Figure 1-1:

► A browser-based client issues an HTTP GET request for a page from a Windows Web server with the .NET Framework installed. The ASP.NET page has a SOAP client proxy—called a Web Reference—that stores a locally cached copy of a WSDL document on a remote server. The page calls a function (WebMethod) defined by the cached WSDL file. For this example, executing the WebMethod sends a SOAP request message to an ASP.NET XML Web service, which returns an XML-encoded SOAP response message to the ASP.NET server. Visual Basic .NET code behind the ASP.NET page transforms the XML content to an XHTML table and sends the XHTML document to the browser.

NOTE
For an example of the preceding type of XML Web service, go to http://www.oakleaf.ws/cfr/. Clicking the Contents link returns an XHTML document that represents the top-level table of contents of the U.S. Code of Federal Regulations.

► Alternatively, the ASP.NET page executes a Transact-SQL (T-SQL) stored procedure that has a similar WSDL file and SOAP wrapper. SQLXML 3.0, which Microsoft also calls SQL Server 2000 Web Services Toolkit for Microsoft .NET, includes a feature that automatically creates WSDL files and SOAP wrappers for SQL Server or MSDE 2000 SELECT stored procedures. Chapter 7’s “Creating XML Web services with Microsoft SQLXML 3.0” section shows you how to use this new feature.
Figure 1-2  ASP.NET and new SOAP-based toolkits for Microsoft SQL Server 2000 and Office XP enable Web servers and Windows applications to consume XML Web services.
Chapter 1: Introducing XML Web Services and the .NET Framework

NOTE

Go to http://www.oakleaf.ws/SQLXML3/ for a demonstration of an SQLXML 3.0 XML Web service. Select the Top 100 option, mark the Amplifiers, DVD Players, and Receivers (Home) check boxes, and click the Retrieve Data button to execute the GetTop100 stored procedure.

A client running a Windows forms application refers to a cached WSDL file and sends a secure HTTPS POST request to the entry function of a COM+ business-process application running under Windows .NET Server’s COM+ Web Services Server. Windows .NET Server lets you SOAP-enable a COM+ component by marking a configuration check box.

An Excel 2002 application that’s been SOAP-enabled by the Office XP Web Services Toolkit searches a public Universal Description, Discovery, and Integration (UDDI) 1.0 registry for an XML Web service that connects to a remote IBM DB2 database of demographic information. After locating the service by a keyword or business name search, the Excel application connects to the WSDL file and executes the specified Enterprise JavaBean function to retrieve SOAP-encoded data for a particular Metropolitan Statistical Area (MSA). Excel VBA code transforms the XML payload of the SOAP message to a named range.

NOTE

Chapter 9’s “Using the Office XP Web Services Toolkit with Excel 2002” section shows you how to use the Toolkit to display formatted financial reports generated by an ASP.NET XML Web service.

With the exception of the Office XP example, all the preceding processes are operating system–agnostic on the client side. The browser and thick client can run on any version of Linux, UNIX, or the MacOS that supports SOAP 1.1 client proxies.

The primary technical and economic benefits of XML Web services derive from mandatory adherence to a set of industry standards. Adopting operating system– and programming language–agnostic W3C specifications lets developers create object packages that are accessible to anyone, anywhere, who needs and is willing to pay for the services they provide. Thus, the potential market for commercial XML Web services is much larger than that for components based on proprietary technologies, such as ActiveX controls, but the business models differ greatly. You purchase a one-time, design-mode license, which includes unlimited run-time distribution, for most commercial ActiveX controls. Providers of XML Web services are more likely to charge a pay-per-use or subscription fee, often with a one-time setup charge.

Standards-based object invocation methods are also important to corporate IT management, because the investment in developing XML Web services survives organizationwide or departmental platform changes and development tool transitions. Depending on internal accounting policies, departments using corporate XML Web services might be assessed setup, usage, or both types of fees.
XML Web Service Examples

Many developers write example XML Web services and make them accessible from SOAP-enabled clients by registration in one of the two public UDDI 1.0 registries run by Microsoft and IBM. Public UDDI registries replicate their data each day, so you need to register your services on only one site. Several sites act as informal registries that provide lists of links to HTML pages that describe the service and provide a link to the developer’s WSDL file. The XMethods site (http://www.xmethods.net/), for example, offers a comprehensive list of sample XML Web services, most of which execute trivial functions, such as area code lookup or units conversion.

The following sections describe Microsoft’s first commercial XML Web services and the nontrivial Visual Basic sample services and SOAP-enabled clients you create as you proceed through the chapters of this book.

.NET Passport

The most controversial member of Microsoft’s commercial XML Web services is .NET Passport for user authentication. (Microsoft added the .NET prefix to the Passport service in late 2001.) .NET Passport provides single-sign-on services for Microsoft’s HotMail, public UDDI registry, and other Web sites that subscribe to the Passport service. Microsoft claimed in late 2001 to have more than 200 million identities stored in their “Passport Vault” and 10,000 servers to process up to 1.5 billion authentications per day.

The next version of Passport .NET, scheduled for implementation sometime in 2002, adds Kerberos v5 authentication to the original cookie-based system and enables organizations to establish their own .NET Passport servers. If your organization meets Microsoft’s security standards and pays a substantial yearly fee, you can “federate” your .NET Passport servers via a cross-domain trust with the Microsoft service. Get further information on .NET Passport and download version 2.0+ of the .NET Passport SDK for Windows 2000/XP/.NET at http://www.microsoft.com/myservices/.

Web privacy organizations and, of course, Microsoft’s competitors question whether Microsoft can be trusted to preserve the sanctity of Passport user-supplied information, such as credit card and demographic data, as well as prevent safecrackers from gaining access to the vault. Sun Microsystems, IBM Corporation, and other major players in the hardware and software business obviously covet Microsoft’s high-profile status in the Web identity market. Sun and 30-some other organizations have formed the Liberty Alliance project to create an “open, federated identity solution,” which translates to “alternative to Passport.” (Go to http://www.projectliberty.org/ for details.) When this book was written, the Liberty Alliance was in the formative stage and collecting membership fees to “… architect, develop, market, and adopt a new, open solution for federated network identity.” Most industry observers doubt that “identity by committee” will overcome the “identity by Microsoft” momentum.

NOTE

Go to http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2830300,00.html for a March 2002 compendium of ZDNet articles on the Passport versus Liberty Alliance battle.
.NET My Services

.NET My Services (formerly codenamed HailStorm) is a suite of Microsoft-hosted XML Web services, most of which are scheduled for commercial deployment in 2002. Version 1.0 of .NET My Services encompasses 15 personal data services for consumers and corporate users, all of which will use .NET Passport’s Kerberos authentication mechanism in the release version. Many of the services duplicate features of Microsoft Exchange, Outlook, and Outlook Express. Following are brief descriptions of the most useful V.1 services:

- **.NET Notification** is the first of the new .NET My Services to go online at eBay, MSN, and a few other third parties. The .NET Notification service uses the Windows or MSN Messenger service to deliver simple notifications to PCs and mobile devices.
- **.NET Calendar** handles time and task management and lets you share your free/busy schedule with others.
- **.NET Contacts** is a server-based address book that’s adaptable to a wide range of devices, such as Pocket and Tablet PCs, Palm-style organizers, and cell phones.
- **.NET Inbox** stores e-mail and voice mail messages on Microsoft’s server instead of your organization’s Exchange server. Some industry analysts equate .NET My Services with Exchange 2000 Server outsourced to Microsoft as the application service provider (ASP).
- **.NET Wallet** is the successor to the Microsoft Wallet (integrated with Passport) and stores credit card information to support the Passport Express Purchase service. Microsoft will probably add some type of micropayment system to .NET Wallet to eliminate credit card transaction fees imposed on pay-per-use Web services.
- **.NET Profile** is a service migrated from the original Passport Profile component, which stores user demographic data, such as names, addresses, telephone numbers, public e-mail addresses, birthday and anniversary dates, and even photographs.

**NOTE**

For a complete list of the version 1.0 services, go to [http://www.microsoft.com/myservices/services/faq.asp](http://www.microsoft.com/myservices/services/faq.asp). You can download the .NET Passport SDK v2.1 from MSDN Downloads; search for “Passport.”


**OakLeaf Systems’ Online Code of Federal Regulations**

The OakLeaf Online Code of Federal Regulations (CFR) ASP.NET sample project is an example of a large-scale content syndication service supported by six Visual Basic .NET

The U.S. Government Printing Office (GPO) and the National Archives and Records Administration (NARA) publish the Electronic CFR (e-CFR), an online version of the CFR, which is updated daily. The e-CFR Web site (http://www.access.gpo.gov/ecfr/) lets you browse the CFR’s 50 titles, 28 subtitles, 375 chapters, 799 subchapters, 7920 parts, and 172,800 sections and appendixes. Sections contain the CFR’s text, which is available in HTML and Standard Generalized Markup Language (SGML) formats. The GPO site also offers full-text search of the sections.

The data source for the CFR site is a 1GB SQL Server 2000 database, which contains a table for each element of the CFR hierarchy. The Titles, Subtitles, Chapters, Subchapters, and Parts tables store tables of contents (TOCs) as well-formed XML documents. TOCs provide the foundation for navigating the CFR hierarchy. The Sections table contains a structured XML document with a header, body, and, when applicable, additional elements for each CFR section. Sections and appendixes range in size from a single short paragraph to very complex, 1+ MB documents. SQL Server 2000’s full-text search service emulates a Web-based search for individual words and complete phrases. The full-text index files for TOCs and section text consume about 150MB of disk space.

TIP

The Home and About pages of the CFR site at http://www.oakleaf.ws/cfr/ offer detailed explanation of the structure of the U.S. CFR, samples of generated XML and XHTML documents, and example WSDL files. The Help page explains the navigation process and has links to CFR sections of general interest.

ASP.NET and the XML Web services transform the XML documents to XHTML for the browser presentation layer. (XHTML is HTML composed as a well-formed XML document so as to be valid for an XML Web service payload.) Alternatively, the XML Web services supply XML documents that subscribers to the syndicated content can transform to HTML, Compact HTML (cHTML), and Wireless Markup Language (WML) to accommodate multiple devices.

Figure 1-3 is a diagram of data flow through the CFR project; the XML Web service components are shaded.

Following are brief descriptions of the major components of the CFR project:

▶ A Visual Basic .NET administrative client application downloads the SGML text of CFR sections and transforms the flat SGML typesetting format to XML documents, which are stored in the CFRSSQL database’s Sections table. The transformation components translate SGML named character references, such as &thnsp;, to their ISO 8879 numeric entity equivalents (&#8201;). Section body text is transformed to a hierarchy of paragraphs in unconventional outline format: (a), (1), (i), (A).
Figure 1-3  This diagram of the OakLeaf Code of Federal Regulations (CFR) project describes the data flow between its components. XML Web services are shaded.
NOTE

Appendix A provides instructions for downloading and attaching the sample CFRSQSQL database to your local instance of SQL Server or MSDE 2000. Appendix B describes how to use the CFRClient.exe application to populate the CFRSQSQL database on your development machine.

- Clients connect to the home page of the CFR Web site, which offers TOC-based navigation, full-text search, and links to Help and About pages. (See Figure 1-4.)
- Clicking the Home page’s CONTENTS link to the default title/subtitle TOC or a TOC element at a lower level begins a typical XML Web service request-response operation.
- The CFRTocWS service’s TransformXMLToc method generates the well-formed (but not validated) XHTML code to display the requested TOC table. (See Figure 1-5.) Alternatively, the ASP.NET page can request the source XML document and locally invoke the XfrmXMLToc method for PC devices.
- The OpenToc.aspx page routes requests for section text to the CFRSectWS service, which obtains the requested XML document from the Sections table and passes it to...
Chapters of the U.S. Code of Federal Regulations (Click a Chapter to Display Its Parts)

<table>
<thead>
<tr>
<th>TITLE</th>
<th>Return to Titles Table of Contents and Links to Search and Help Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>TITLE 7</td>
<td>AGRICULTURE</td>
</tr>
<tr>
<td>Subtitle B</td>
<td>Regulations of the Department of Agriculture</td>
</tr>
<tr>
<td>Chapter I</td>
<td>Agricultural Marketing Service (Standards, Inspections, Marketing Practices), Department of Agriculture (Parts 27 - 299)</td>
</tr>
<tr>
<td>SUBCHAPTER A</td>
<td>COMMODITY STANDARDS AND STANDARD CONTAINER REGULATIONS</td>
</tr>
<tr>
<td>SUBCHAPTER C</td>
<td>REGULATIONS AND STANDARDS UNDER THE AGRICULTURAL MARKETING ACT OF 1946</td>
</tr>
<tr>
<td>SUBCHAPTER D</td>
<td>EXPORT AND DOMESTIC CONSUMPTION PROGRAMS</td>
</tr>
<tr>
<td>SUBCHAPTER E</td>
<td>COMMODITY LABORATORY TESTING PROGRAMS</td>
</tr>
<tr>
<td>SUBCHAPTER F</td>
<td>NAVAL STORES</td>
</tr>
<tr>
<td>SUBCHAPTER K</td>
<td>FEDERAL SEED ACT</td>
</tr>
<tr>
<td>Chapter II</td>
<td>Food and Consumer Service, Department of Agriculture (Parts 210 - 299)</td>
</tr>
<tr>
<td>SUBCHAPTER A</td>
<td>CHILD NUTRITION PROGRAMS</td>
</tr>
<tr>
<td>SUBCHAPTER B</td>
<td>GENERAL REGULATIONS AND POLICIES - FOOD DISTRIBUTION</td>
</tr>
<tr>
<td>SUBCHAPTER C</td>
<td>FOOD STAMP AND FOOD DISTRIBUTION PROGRAM</td>
</tr>
<tr>
<td>SUBCHAPTER D</td>
<td>GENERAL REGULATIONS</td>
</tr>
<tr>
<td>Chapter III</td>
<td>Animal and Plant Health Inspection Service, Department of Agriculture (Parts 300 - 2003)</td>
</tr>
</tbody>
</table>

Figure 1-5 Navigation by TOC passes the TocId attribute value (in this case, ti7stB for title 7, subtitle B) to the CFRTocWS service’s GetTocByToc function.

the XfrmSectWS service to format the header and text as XHTML. (See Figure 1-6.) XML paragraph subelements determine indentation of nested paragraphs.

The Search.aspx page (see Figure 1-7) contains a table-based form that lets you enter a search expression, restrict the search to TOCs or section text, specify all CFR titles or a single title, determine the depth of TOC searches, and limit the number of hits returned. Figure 1-8 shows part of the results returned by a search with the default “satellite NEAR broadcast” search term.

NOTE

Microsoft and IBM UDDI 1.0 registries contain fully documented entries for the three XML Web services that support the OakLeaf CFR application. Go to http://uddi.microsoft.com/ and type oakleaf in the Search text box. Click the OakLeaf Systems link to open the Business Details page, which has links to the individual services.
The ASP.NET version of the CFR project is the result of a three-stage upgrade from the original Visual Basic 6.0 components and conventional ASP technology. The CFR-COM version uses ActiveX DLLs. COM components of the CFR-SOAP version are upgraded to XML Web services with the SOAP Toolkit 2.0. The next step was CFR-ASPX, which uses ASP.NET in ASP compatibility mode to consume the COM-based XML Web services. You can access the original and the three upgrade versions of the project from the default page at http://www.oakleaf.ws. There is little difference in the appearance or performance of the four versions.

Chapter 9’s “Drilling Down with DHTML Navigation” section explains how the CFR client application works. The chapter’s downloadable sample code includes the complete ASP.NET code and other files for the client side. You can install the client on your test machine and consume OakLeaf’s remote CFRTocWS, CFRSectWS, and CFRSearchWS XML Web services.
Chapter 1: Introducing XML Web Services and the .NET Framework

Writing the Visual Basic data delivery components, designing the .asp pages, and upgrading the project to XML Web services and, eventually, to ASP.NET and ADO.NET was a piece of cake compared to the effort involved in writing the code to translate the GPO's SGML into well-formed XML for populating the database. SGML is intended primarily for typesetting and doesn't adhere to XML's strict requirements for well-formedness, such as matching opening and closing element tags. The GPO has established a very detailed Document Type Definition (CFRDOC.DTD) for a future XML version of the CFR, which you can review at http://www.access.gpo.gov/ecfr/cfrxml/rdtd.txt.

OakLeaf Consumer Electronics’ B2C and B2B Project

The OakLeaf Consumer Electronics (OCE) project is an example of a Visual Basic .NET/ASP.NET B2C retail application that incorporates a B2B supply-chain system. The OCE sample project is typical of distributed applications that use vendors’ XML.
Web services to perform credit card transactions, manage shipping operations, replenish inventory, and fulfill nonstock or special orders directly from vendors’ inventory. Figure 1-9 illustrates the interconnections between OCE customers, service providers, and suppliers. In B2B-speak, these organizations are called “business partners.” Partnership infers joint and several liability of partners, so “vendors” or “suppliers” and “service providers” are more accurate terms for third-party participants in sales transactions.

You build the Visual Basic .NET XML Web services and consuming client applications for the OCE project in Chapters 2 through 10 of this book. The databases have a sufficient number of records to determine relative performance of different data access methodologies. Following are brief descriptions of the databases and processes involved in the OCE project:

- The Customers/Orders database contains a Customers table with 25,344 customer names, postal addresses, phone numbers, e-mail addresses, and account passwords.
- The Products/Inventory database contains a Products table listing 695 consumer electronics products from 26 manufacturers. The products are purchased from three local distributors (Alpha Electronics, Beta Electronics, and Gamma Electronics), each of which offers automated order processing implemented as XML Web services.
OmegaBank's Authorization database contains 38,188 fictitious American Express, Discover, MasterCharge, and Visa credit cards for OakLeaf's 25,344 customers. The record for each card includes ChargeLimit, Charges, Holds, and Available (credit) fields.

Figure 1-9 The OakLeaf Consumer Electronics project emulates a large-scale distributed supply-chain application that uses third-party XML Web services for transactions with suppliers and service providers.

- OmegaBank's Authorization database contains 38,188 fictitious American Express, Discover, MasterCharge, and Visa credit cards for OakLeaf's 25,344 customers. The record for each card includes ChargeLimit, Charges, Holds, and Available (credit) fields.
NOTE

All customer and credit card information in the databases is fictitious. The products offered represent actual consumer electronic devices from the manufacturers specified in the database, but price and cost data don’t represent actual retail and distributor prices.

► When a customer places an order, the ASP.NET project verifies the customer’s credit card and available credit, checks product inventory (locally or at the distributor), and computes and adds the shipping charge and sales tax, where applicable. If the products are in inventory, the initial transaction places a hold for the total amount of the order against the credit card, classifies the products ordered as committed inventory, issues a ship order to the warehouse or distributor, and sends an acknowledgment of the order to the customer as an e-mail message or .NET Notification.

► Upon shipment of the goods, the final transaction removes the hold, debits the customer’s credit card, decrements the products’ inventory level, and sends another message or .NET Notification to customers with shipment and tracking information.

Each step in the initial and final transactions is subject to business rules implemented by Visual Basic .NET Windows form and Web Form clients. For example, the initial transaction aborts if the customer’s credit card number is invalid, expired, or canceled, or the customer has insufficient credit availability for the purchase.

Figure 1-10 shows the ASP.NET Web Form version of the client consumer for the OmegaBank, AlphaDist, BetaDist, and GammaDist XML Web services. The ALPH entry in the Dist column for the first of the two line items in the figure indicates that shipment of the item has been outsourced to Alpha Distributors, Inc. To test drive the Web Form version of the OCE_Client application, go to http://www.oakleaf.ws/oce/ and click the ASP.NET page’s Add One Order button repeatedly to add orders to the remote OCE and distributor databases.

NOTE

Technically, OmegaBank isn’t an XML Web service because it returns a cookie-like text string, not a well-formed XML response document.

Like the CFR XML Web services, the UDDI 1.0 production registries have entries for the three distributor services. To open and test the WSDL document for a distributor XML Web service, do this:

1. Go to http://uddi.microsoft.com and search for Alpha Dist. Alternatively, click the Advanced Search link, type oakleaf in the Search For text box, and choose Business Identifier from the list to display a page that has links to all four services registered by OakLeaf Systems.

2. Click the link to the Business Identifier page, and then click a Name link in the Services list to open the Service Details page.

3. Click the Access Point link to open the service’s WSDL page, AlphaDist for this example, which has HTML-formatted service documentation. (See Figure 1-11.)
Chapter 1: Introducing XML Web Services and the .NET Framework

4. Click the Service Description link to display the WSDL document.

5. Return to the WSDL page and click the Check Stock link to open a test form that uses the HTTP GET protocol to return an XML response document generated by the CheckStock method.

6. Type the five values shown in boldface into the test form’s corresponding text boxes.

7. Click Invoke to display the XML response document. (See Figure 1-12.)

NOTE

Chapter 11 shows you how to navigate the UDDI 1.0 public registries and register your XML Web services.

The Role of XML Web Service Providers

Remote hosting of high-volume public Web sites by major Internet service providers, such as IBM, AT&T, and Exodus, has been common since the late 1990s. The usual approach to remote hosting has been co-location, where the Web site owner rents space (a cage) and
Internet bandwidth from the provider. Security issues and the cost of Web server management have created a market for complete Internet (and often intranet) infrastructure management by independent firms called application service providers (ASPs).

NOTE

The term ASP originally applied to organizations that outsourced server-based applications, such as Microsoft Exchange, customer relationship management programs, e-mail systems, and the like, on the ASP’s hardware. Outsourcing a Web server farm, regardless of who owns the equipment, is similar to outsourcing any other application, so ASP has grown to encompass services that manage customers’ Internet, intranet, or both operations.

Third-party XML Web service providers are ASPs that specialize in hosting XML Web services and, in many cases, their underlying databases. Some WSPs offer support for multiple platforms, such as Linux, Windows, and various UNIX flavors, and rent space on Oracle, IBM, Sybase, and other database servers. Others specialize in a single technology, such as Microsoft’s .NET Framework and SQL Server 2000. The advantages of outsourcing...
XML Web services, and to a lesser extent, data storage, are similar to those for outsourcing Web farms—availability, reliability, scalability, and, most importantly, security.

Mike Amundsen’s Eraserver.net (http://www.eraserver.net) is one of the first WSPs to specialize in hosting .NET XML Web services. Hosting XML Web services on Eraserver.net starts at $9.95 per month; add $10.00 for 10MB of SQL Server database space and another $10.00 for domain hosting. For a list of hosting services that support the .NET Framework and ASP.NET XML Web services, go to http://www.gotdotnet.com/resourcecenter/resource_center.aspx, and select Web Hosting from the Type list. Few, if any, of these hosting firms, however, provide availability and performance guarantees.

Enterprise-level hosting services that offer availability and performance guarantees do so through service-level agreements (SLAs). An SLA specifies guaranteed availability, usually expressed as an uptime percentage—typically 99.9 percent (“three nines”), 99.99 percent (“four nines”) or 99.999 percent (“five nines”). Four- or five-nine availability usually requires server clusters and multiple Web server front ends. Hardware-based firewalls provide security against intrusion and site hacking. If the service fails to meet the guaranteed availability level, the customer receives a partial or complete refund for a specified period of time. Charges for enterprise-level hosting services with stringent SLAs start at $10,000 per month or more. Initial hardware and setup charges, which are becoming more common as enterprise-level ASPs burn through their original venture capital, often exceed $100,000.

**XML Web Service Pitfalls**

One potential drawback of XML Web services is versioning problems. A primary tenet of COM component design is that interfaces are immutable. That is, after you’ve created a production object, never alter the name or arguments of its public methods. Fixing internal

---

Figure 1-12 The WSDL page’s test form returns the response document of XML Web services that support HTTP GET requests.
bugs and recompiling a Visual Basic component, however, is allowable if you compile with the appropriate version compatibility option. Otherwise, you must recompile and redeploy every client app to accommodate the updated version.

When your application integration projects involve external XML Web services, which are under the control of another organization and its developers, you can’t count on the outside developers to adhere to the immutable component interface rule. Beware of XML Web services that carry “subject to change without notice” in their licensing or subscription agreements. A minor change to a single XML Web service can bring your entire enterprise application integration system to an abrupt halt.

Internet latency and reliability issues are another major concern when your distributed system relies on external XML Web services. Even if the provider of the XML Web service offers a four-nines SLA, the uptime applies only to the remote site, not your ability to connect to it from your server. Quality of Service (QoS) contracts for both endpoints (servers, in this case) can minimize latency over a fixed route, but provisioning QoS is costly to implement. The latency of Internet connections, even with a high QoS level, requires that the database supporting the XML Web service be either local to the site or connected by a dedicated, high-speed (T-3 or better) connection.

Stringent security and snappy performance are mutually exclusive objectives. Until the W3C establishes final security standards for SOAP messages, SSL versions 2.0 and 3.0—which implement the Public Key Infrastructure (PKI), and HTTPS as the transport protocol—are the best (and often the only) choice to maintain security with an Internet connection. Endpoint authentication and message encryption add a substantial amount of overhead to XML Web service connections. Tests with typical request-response operations against moderate-sized databases indicate that HTTPS transmission time increases by about 50 percent of the HTTP round-trip time. There’s no guarantee that proposed IBM/Microsoft/VeriSign WS-Security SOAP extension or the W3C’s XML Signature and XML Encryption standards will be significantly faster or more secure than traditional SSL and HTTPS.

Another issue developers often raise when discussing the merits of XML and its dialects is the wire overhead added by XML’s requirement for human-readable text, long-winded Uniform Resource Identifiers (URIs) for namespace declarations, and other baggage carried by XML-derived languages. Fortunately, HTTP 1.1 text-compression algorithms for readable text and, especially, repetitive tag names and attribute values are very efficient. Performance penalties when substituting XML Web services for DCOM components, for example, are insignificant in most cases. Increases in LAN and WAN performance during XML Web services’ commercial gestation period will more than overcome the wire overhead imposed by WSDL/SOAP encoding.

Core XML Web Service “Standards”

XML Web services rely on the W3C specifications (couch as “recommendations”) for XML and XSD, but SOAP currently has the status of a W3C note. WSDL, which isn’t mentioned in the SOAP note, is a W3C note that’s derived from an amalgam of IBM’s Network Accessible Services Specification Language (NASSL), DevelopMentor’s Component Description Language (CDL), and Microsoft’s Service Description Language (SDL) and Service Contract
Language (SCL). UDDI is an invention of the “committed industry leaders” who founded UDDI.org, and doesn’t have even W3C note status. The amount of press coverage devoted to a new and largely unproven technology based on pseudostandards is surprising, to say the least. Publicity alone has made SOAP a de facto industry standard.

NOTE

Version 1.1 of SOAP and WSDL were current when this book was written. W3C’s Web Services Activity, which incorporated the XML Protocol Activity in January 2002, is responsible for establishing a SOAP 1.2 recommendation. You can read the current working draft of the two-part SOAP 1.2 specification at http://www.w3.org/TR/soap12-part1/ and http://www.w3.org/TR/soap12-part2/.

This book doesn’t waste pages with an “XML primer” or other introductory XML content. It’s assumed that you’re at least familiar with XML fundamentals or are willing to gain your XML vocabulary by immersion as you progress through this book. XSD coverage is limited to those elements applicable to WSDL and SOAP request/response documents. The following sections provide an introduction to the XML-based pseudostandards, which Microsoft calls “baseline XML Web service standards,” and the W3C recommendations that support or complement XML Web services.

NOTE

Chapter 2 covers the SOAP and WSDL 1.1 specifications in detail. Chapter 10’s “Validating XML Request and Response Messages with XSD Schemas” and “Applying Element Datatype Checking to XML Web Services” sections describe how to validate SOAP request and response documents.

Simple Object Access Protocol (SOAP)

SOAP is a specification for a wire protocol designed primarily for invoking methods of objects with a combination of XML and HTTP as the invocation mechanism. The SOAP specification describes an XML vocabulary for representing methods, parameters, return values, and errors (exceptions). SOAP is a standards-based substitute for proprietary RPC protocols, such as DCOM and Internet-InterORB Protocol (IIOP), which aren’t well suited to transmission via the public Internet. One of the primary benefits of SOAP is use of HTTP 1.0+ as its transport; all firewalls allow HTTP connections on TCP port 80. Although SOAP supports other transports, use of HTTP is the “Open Sesame” that let the XML Web services genie out of the bottle.

The origins of SOAP extend back to early 1998, when Dave Winer of Userland Software (http://www.userland.com/), Don Box, and members of Microsoft’s COM+ (then COM/MTS) group drafted an early version of the current specification. According to Don Box (see http://www.xml.com/pub/a/2001/04/04/soap.html), who’s now an architect in the Microsoft .NET Developer and Platform Evangelism group, disagreements between Microsoft’s XML and COM+ groups caused a hiatus in the specification development process. To fill the gap, Dave Winer published in April 1998 an XML-RPC specification (http://www.xmlrpc.com/) from a subset of the original SOAP proposal. XML-RPC quickly gained implementations in all major Web-based programming languages and platforms.
Microsoft’s subsequent decision to “embrace and extend” XML, which is similar in scope to the firm’s December 1995 “embrace and extend” manifesto for Internet-enabling Windows, sped up the authoring process, and the SOAP version 1.0 specification shipped in late 1999. Version 1.1, which gained contributors from IBM and its Lotus subsidiary, was published as a W3C note dated May 8, 2000 (http://www.w3.org/TR/SOAP/). Version 1.1 incorporated some minor technical clarifications and changes; the only new feature added in SOAP 1.1 was the optional `actor` attribute.

A SOAP message is a one-way transmission between endpoints, called `sender` and `receiver`. A SOAP request message usually results in a SOAP response message delivered as an HTTP response to the sender on the request connection. SOAP messages consist of the following three basic elements:

- `<SOAP-ENV:Envelope>` is the top element of all SOAP messages and is mandatory.
- `<SOAP-ENV:Header>` is an optional element. If present, this element must be the first child element of `<SOAP-ENV:Envelope>`. Header elements enable SOAP messaging to handle extensions, such as WS-Security or WS-Routing, which are described in the later “Microsoft Global XML Web Service Architecture Extensions” section.
- `<SOAP-ENV:Body>` is a mandatory element that contains the payload, which must be a well-formed XML document, and an optional single instance of a `<SOAP-ENV:Fault>` element for handling exceptions. Today, the most common payload for a request message is an object method call; the response message’s payload is the result of the method’s execution, such as a simple string or numeric value. As XML Web services mature, use of XML documents for request and response messages will become more common.

The following two sections illustrate a real-life (nontrivial) pair of SOAP request and response messages. The samples are for a Visual Basic 6.0 ActiveX DLL with a SOAP wrapper applied by the Microsoft SOAP Toolkit 2.0. The Toolkit’s Trace Utility (`MsSoapT.exe`) captured the two messages that result from clicking the Search Now button of the CFR project’s `.Search.aspx` page.

**NOTE**

You might question why the examples of this chapter use Visual Basic 6.0 components instead of ASP.NET Web Service projects. The answer is that most Visual Basic developers will cut their XML Web services eyeteeth with existing ActiveX DLLs and the SOAP Toolkit. Even if your DLL doesn’t emit well-formed XML, it’s reasonably easy to modify the DLL code to send the result as unparsed `CDATA` or base64-encoded characters.

**SOAP Request Message Example**

The following document is a sample SOAP message that requests the XML source document for the default full-text search against the CFRSQL database shown in Figure 1-7, except for a 10-hit (rather than 100-hit) limit. The `GetSearchResults` method of the `CFRSearch` object has nine arguments: `strConnect`, `strSearch`, `intMaxHits`, `intMinLevel`, `intMaxLevel`, `strTitleID`, `lngTOCItems`, `lngSectItems`, and `strSearchType`.
For this example, the value of strSearchType is XML; substituting XHTML returns the corresponding XHTML document for browser presentation.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAPSDK1:GetSearchResults
      xmlns:SOAPSDK1="http://oakleaf.ws/message/"
      xmlns:SOAPSDK2="http://oakleaf.ws/message/">
      <strConnect>
        Provider=SQLOLEDB;Data Source=OAKLEAF-MS9;
        Initial Catalog=CFRSQL;UID=CFRUser;PWD=charon123
      </strConnect>
      <strSearch>
        satellite NEAR broadcast
      </strSearch>
      <intMaxHits>
        10
      </intMaxHits>
      <intMinLevel>
        0
      </intMinLevel>
      <intMaxLevel>
        7
      </intMaxLevel>
      <strTitleID />
      <lngTOCItems>
        9179
      </lngTOCItems>
      <lngSectItems>
        172801
      </lngSectItems>
      <strSearchType>
        XML
      </strSearchType>
    </SOAPSDK1:GetSearchResults>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The assignment of the method name and its argument values to XML elements is straightforward and easy for any Visual Basic developer to interpret. In this example, the argument values are simple strings and numeric types, which don’t require datatype definitions in the message. Chapters 2 and 10 deal with these and more complex datatypes, such as structures and arrays. Chapter 3 shows you how to use the Microsoft SOAP Toolkit 2.0.
NOTE

When this book was written, the SOAP Toolkit 3.0 was in the prebeta stage. Version 3.0 of the Toolkit should be available as a beta or release version in mid-2002.

SOAP Response Message Example

Sending the preceding request message results in the following SOAP response message, which contains a well-formed XML document as its <SOAP-ENV:Body> payload. The SOAP wrapper appends Response to the name of the method and delivers the document within the <Result> subelement. The <Header> subelements generate the first four rows of the table shown in Figure 1-8; the <ColHeads> subelement specifies the number of columns and column names for the search hits. Result elements at the <Sections> level include  id attribute values for navigation and populating the ten result rows.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope
   SOAP-ENV:encodingStyle=
     "http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAPSDK1:GetSearchResultsResponse
      xmlns:SOAPSDK1="http://oakleaf.ws/message/"
    >
      <Result>
        <?xml version="1.0" encoding = "UTF-8" ?>
        <CFRSearch xmlns:CFRSections=
          "http://www.oakmont.org/CFR/Search">
          <Header>
            Full-Text Search of the XML Version of the
            Electronic CFR (eCFR Beta)
          </Header>
          <Level>
            Starting at the Sections level in the eCFR
            Table of Contents
          </Level>
          <Search>
            Searching for [satellite NEAR broadcast] with
            a limit of 10 elements
          </Search>
          <Result>
            Your search returned 10 elements from 181,980
            records in 0.12 seconds
          </Result>
          <ColHeads>
            4:Title:Part:Section:Section Name
          </ColHeads>
```
Chapter 1: Introducing XML Web Services and the .NET Framework

102 - Definitions.

2 - Definitions.

4 - Types of projects and broadcast priorities.

3 - What definitions apply to this part?

11 - Satellite carrier statements of account covering statutory licenses for secondary transmissions for private home viewing.

2 - Purpose of Copyright Arbitration Royalty Panels.

283 - Authority delegated.

51 - Functions of the Bureau.

4000 - Restrictions impairing reception of television broadcast signals, direct broadcast satellite services, or multichannel multipoint distribution services and restrictions impairing reception or transmission of fixed wireless communications signals.

33 - EAS Decoder.
SOAP Message Attachments

Conventional SOAP messages consist of a well-formed XML document; if the message isn’t well formed, XML parsers reject the entire envelope. In many cases, it’s desirable to include non-text attachments, such as fax and bitmap images, with the SOAP payload. Microsoft’s SOAP Messages with Attachments specification defines “SOAP message packages” that contain a primary (XML) document and attachments of the MIME Multipart/Related media type (IETF RFC 2557, http://www.ietf.org/rfc/rfc2557.txt). You can read Microsoft’s W3C note at http://www.w3.org/TR/SOAP-attachments.

NOTE

SOAP Messages with Attachments isn’t a “core” XML Web services protocol, but it shares W3C note status with SOAP and WSDL. An alternative to base64 encoding non-XML content or incorporating a MIME attachment is to provide a URL for the attachment and rely on the consuming application to retrieve the binary data.

In May 2001, Microsoft published a specification for Direct Internet Message Encapsulation (DIME), which you can read at http://www.gotdotnet.com/team/xml_wsspecs/dime/dime.htm. The specification’s authors describe DIME as a “lightweight, binary encapsulation format that can be used to encapsulate multiple application-defined entities or payloads of arbitrary type and size into a single message construct.” The DIME specification is intended to improve the performance of SOAP messages with attachments by eliminating the need to parse an entire MIME enclosure to determine if and how the enclosure applies to the message. Microsoft submitted DIME and Encapsulating SOAP in DIME as Internet drafts to the IETF on February 1, 2002. You can read both drafts at http://gotdotnet.com/team/xml_wsspecs/dime/default.aspx.

Web Services Description Language (WSDL)

WSDL (commonly pronounced “whizdle”) is a standardized metadata language—developed jointly by Microsoft and IBM—for defining XML Web services in terms of service endpoints (ports) to which clients connect with a specified protocol (binding), object invocation (operation), and message format (SOAP). You can read the WSDL version 1.1 specification, which was current when this book was written, at http://www.w3.org/TR/wsdl.

The WSDL 1.1 specification provides examples only for SOAP 1.1, HTTP GET/POST, and MIME. Given a sufficiently detailed WSDL file, an application can automatically generate a SOAP client proxy from the WSDL document. The SOAP Toolkit’s WSDL Generator utility (Wsdllgen.exe) generates WSDL 1.1 files named WebServiceName.wsdl.
When you create an ASP.NET Web Service project with the `<WebService>` class attribute, ASP.NET creates a WebServiceName.asmx page that contains a WSDL document, not a freestanding .wsdl file.

As you can see from the following Toolkit-generated CFRSearchWS.wsdl file for the CFR project’s full-text search page, writing a custom WSDL file in Notepad isn’t a trivial task:

```xml
<?xml version='1.0' encoding='UTF-8' ?>
<!-- Generated 10/08/01 by Microsoft SOAP Toolkit WSDL File Generator, Version 1.02.813.0 -->
<definitions name='CFRSearchWS' targetNamespace='http://oakleaf.ws/wsdl/'
    xmlns:wsdlns='http://oakleaf.ws/wsdl/'
    xmlns:typens='http://oakleaf.ws/type'
    xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
    xmlns:xsd='http://www.w3.org/2001/XMLSchema'
    xmlns:stk='http://schemas.microsoft.com/soap-toolkit/wsdl-extension'
    xmlns='http://schemas.xmlsoap.org/wsdl/'>
    <types>
        <schema targetNamespace='http://oakleaf.ws/type'
            xmlns='http://www.w3.org/2001/XMLSchema'
            xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'
            xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
            elementFormDefault='qualified'>
        </schema>
    </types>
    <message name='CFRSearch.GetSearchResults'>
        <part name='strConnect' type='xsd:string'/>
        <part name='strSearch' type='xsd:string'/>
        <part name='intMaxHits' type='xsd:short'/>
        <part name='intMinLevel' type='xsd:short'/>
        <part name='intMaxLevel' type='xsd:short'/>
        <part name='strTitleID' type='xsd:string'/>
        <part name='lngTOCItems' type='xsd:int'/>
        <part name='lngSectItems' type='xsd:int'/>
        <part name='strSearchType' type='xsd:string'/>
    </message>
    <message name='CFRSearch.GetSearchResultsResponse'>
        <part name='Result' type='xsd:string'/>
    </message>
    <portType name='CFRSearchSoapPort'>
        <operation name='GetSearchResults'
            parameterOrder='strConnect strSearch intMaxHits
            intMinLevel intMaxLevel strTitleID lngTOCItems
            lngSectItems strSearchType'>
...
Visual Basic .NET XML Web Services Developer's Guide

```xml
<input message='wsdlns:CFRSearch.GetSearchResults' />
<output message='wsdlns:CFRSearch.GetSearchResultsResponse' />
</operation>

<binding name='CFRSearchSoapBinding' type='wsdlns:CFRSearchSoapPort' >
    <stk:binding preferredEncoding='UTF-8'/>
    <soap:binding style='rpc'
                 transport='http://schemas.xmlsoap.org/soap/http'/>
    <operation name='GetSearchResults'>
        <soap:operation soapAction='http://oakleaf.ws/action/CFRSearch.GetSearchResults'/>
        <input>
            <soap:body use='encoded' namespace='http://oakleaf.ws/message/'
                      encodingStyle='http://schemas.xmlsoap.org/soap/encoding'/ />
        </input>
        <output>
            <soap:body use='encoded' namespace='http://oakleaf.ws/message/'
                      encodingStyle='http://schemas.xmlsoap.org/soap/encoding'/ />
        </output>
    </operation>
</binding>

<service name='CFRSearchWS'>
    <port name='CFRSearchSoapPort' binding='wsdlns:CFRSearchSoapBinding'>
        <soap:address location='http://66.123.163.243/cfr/CFRSearchWS.WSDL'/>
    </port>
</service>
</definitions>

NOTE

This book often uses IP addresses instead of domain names to invoke XML Web services. The use of hard-coded IP addresses improves service performance by eliminating DNS lookup operations. The oakleaf.ws domain provides a unique URI for the targetNamespace attribute of the top <definitions> element. The .ws (Western Samoa) top-level domain will undoubtedly become popular for sites offering commercial Web services.

If you skip the laundry list of namespace declaration attributes at the beginning of the sample WSDL document, it’s easy to associate WSDL elements with the two SOAP messages of the two preceding sections. The <portType> section’s <input> element includes the programmatic ID (ProgID) registered for the ActiveX DLL. (A registered copy of the DLL or, less commonly, a DCOM proxy, must exist on the machine that hosts the WSDL file.) A <part> element specifies each parameter’s name and datatype; parameters require XSD datatype identifiers, many of which correspond to Visual Basic .NET’s elementary datatypes. The later “XML Schema Definition (XSD) Language” section has more information on XSD datatypes. The <portType> element’s <operation> section defines <input> (request)
Chapter 1: Introducing XML Web Services and the .NET Framework

and <output> (response) message names. The <binding> element specifies the wire protocol (HTTP), and <service> adds the URL to which clients connect to read the WSDL file and execute the method.

Web Services Meta Language (WSML) defines a Microsoft proprietary XML syntax required for SOAP-enabled COM components. When you create a WSDL file, Wsdlgen.exe writes the corresponding WSML file and uses the information it contains to generate the final WSDL file. ASP.NET Web Service projects don’t need a WSML file.

```xml
<?xml version='1.0' encoding='UTF-8' ?>
<!-- Generated 10/08/01 by Microsoft SOAP Toolkit WSDL File Generator, Version 1.02.813.0 -->
<servicemapping name='CFRSearchWS'>
  <service name='CFRSearchWS'>
    <using PROGID='CCFRSearch.CFRSearch' cachable='0' ID='CFRSearchObject' />
    <port name='CFRSearchSoapPort'>
      <operation name='GetSearchResults'>
        <execute uses='CFRSearchObject' method='GetSearchResults' dispID='1610809344'>
          <parameter callIndex='1' name='strConnect' elementName='strConnect' />
          <parameter callIndex='2' name='strSearch' elementName='strSearch' />
          <parameter callIndex='3' name='intMaxHits' elementName='intMaxHits' />
          <parameter callIndex='4' name='intMinLevel' elementName='intMinLevel' />
          <parameter callIndex='5' name='intMaxLevel' elementName='intMaxLevel' />
          <parameter callIndex='6' name='strTitleID' elementName='strTitleID' />
          <parameter callIndex='7' name='lngTOCItems' elementName='lngTOCItems' />
          <parameter callIndex='8' name='lngSectItems' elementName='lngSectItems' />
          <parameter callIndex='9' name='strSearchType' elementName='strSearchType' />
          <parameter callIndex='-1' name='retval' elementName='Result' />
        </execute>
      </operation>
    </port>
  </service>
</servicemapping>
```
XML Schema Definition (XSD) Language

All XML documents that participate in XML Web services must be well formed. Being well formed means that the document complies with the grammar and syntax rules of the W3C XML 1.0 specification. XML parsers, such as MSXML v3+, report errors when opening a malformed XML document. MSXML v3+ has a parseError object that you can interrogate to determine whether an error has occurred, and, if so, what caused the error. Being well formed, however, doesn’t mean the XML document is valid for its intended purpose. Validation is the process of assuring that the document’s contents conform to a particular structure and a specific set of rules for the document’s class.

XML is a derivative of the SGML document formatting language, which uses Data Type Definition (DTD) documents for validation. Thus, it’s not surprising that the first validation method for XML documents also used DTDs. DTDs are intended primarily for describing text-based documents, are very complex, and are exceedingly difficult to write correctly. As developers adopted XML for exchanging data between database servers and applications, it became clear that DTDs weren’t suited to validating documents that contained datatypes other than text strings. Another disadvantage of DTDs is that they aren’t written in XML, so they’re difficult to read and parse.

XML-Data was the first attempt to replace DTDs with schemas for defining and documenting object classes within XML documents. Microsoft was the primary sponsor of the XML-Data specification, which was published as a W3C note in 1998 (http://www.w3.org/TR/1998/NOTE-XML-data/). XML-Data provided XML <elementType> declarations having an optional <description> element for documentation, and <key> elements to support one-to-many relationships. XML-Data also defined datatype attributes, such as dt:dt="int" and dt:dt="string". Microsoft adopted a subset of XML-Data—named XML-Data Reduced (XDR)—as the standard schema definition language for SQL Server 2000’s XML features, BizTalk server, and other XML-based programs. The MSXML parser released with IE 5.0 handles document validation by XDR schemas.

After a long gestation period in the XML Schema Working Group, W3C published the final recommendations for XML Schema in May 2001. The specification consists of three documents: XML Schema Part 0: Primer (http://www.w3.org/TR/xmlschema-0/), XML Schema Part 1: Structures (http://www.w3.org/TR/xmlschema-1/), and XML Schema Part 2: Datatypes (http://www.w3.org/TR/xmlschema-2/). Microsoft has adopted XSD in all newly released products; XDR is supported by the .NET Framework but is relegated to “legacy” status. MSXML 4.0 Core Services, released in September 2001, validates documents with XSD or XDR.

SQLXML 2.0+ supports both XSD and XDR schemas for XPath queries, and you can use the XMLSchemaCollection member of the .NET Framework’s System.Xml.Schema namespace to validate XML documents against XSD or XDR schemas.
The Datatypes section of the XML Schema specification is of most interest to XML Web service developers, because it defines the simple datatypes you can assign to method arguments and elements of SOAP response documents. Following are the three primary classes of XML Schema datatypes:

- **Built-in primitive datatypes** include string, decimal, float, double, boolean, base64Binary, and hexBinary, many datatypes for date, time, and duration, and miscellaneous types, such as anyURI, QName, and NOTATION.

- **Built-in derived datatypes** represent subclasses of primitive datatypes defined by values of constraining facets that restrict a datatype’s value. For example, integer is derived from decimal with the fractionDigits facet value set to 0. Each primitive datatype has a collection of valid constraining facets. Subclasses of integer include long, int, short, byte, negativeInteger, nonPositiveInteger, positiveInteger, nonNegativeInteger, unsignedLong, unsignedInt, unsignedShort, and unsignedByte. Both types of built-in datatypes are called atomic.

- **User-derived datatypes** are made up of atomic datatypes and include list and union datatypes. The most common application for simple user-derived datatypes is data validation by <pattern> elements that use regular expressions to define valid data. Individual list values are separated by whitespace, so list elements can’t contain spaces or newline characters.

The preceding three categories of simple datatypes are capable of handling most method arguments and values returned from relational databases. Complex type definitions, which Part 1: Structures describes, support objects representing business documents, such as purchase orders, or rows of relational data.

Section 5.2 of the SOAP 1.1 specification adopts all built-in datatypes of XSD, plus XSD enumerations. The SOAP specification additionally defines two compound datatypes: Struct, which is similar to an XSD complex type, and Array. The Array datatype supports multidimensional and sparse arrays, which are useful for adding or updating records of tables having one-to-many relationships. Don Box observes that the SOAP 1.0 specification would have been only three or four pages long if the XSD specification had been completed prior to finalizing the SOAP W3C note.
Universal Description, Discovery, and Integration (UDDI)

UDDI is the result of a joint effort by Microsoft, IBM, and others to provide a means for potential users to locate XML Web services that perform a particular function. UDDI consists of specifications for data structures, a programmer’s API, and replication, together with XSD schemas for each type of document. You can download version 2.0 of the complete set of UDDI specifications from http://www.uddi.org/specification.html. The UDDI.org site also offers white papers and technical/executive backgrounders.

NOTE
Unlike WSDL, there’s no generally accepted pronunciation for UDDI. Some developers say “ooh-dee” and others stick with more formal “u-d-d-i.”

Microsoft and IBM operate public UDDI version 1.0 registries, which use replication to keep the individual sites in sync. UDDI registrars assist users with registering their services in one of the public registries and searching for third-party services. (The Microsoft registry requires .NET Passport authentication). Chapter 11 shows you how to register and search for services from your browser or by using the Microsoft UDDI Toolkit, which you can download from http://www.microsoft.com/downloads/release.asp?ReleaseID=35982.

As mentioned in the earlier “XML Web Service Examples” section, you can search Microsoft’s UDDI 1.0 registry for a specific company name or by several different industry classification schemes (http://uddi.microsoft.com/search.aspx). Most UDDI registrations are blatant advertisements and have no XML Web service registrations. Controlling spam in the registries presents a major problem for UDDI operators. One possibility is to drop registrations that don’t point to a valid WSDL file within 30 days of adding the company information.

NOTE
UDDI 2.0 was in the beta testing stage when this book was written, and several additional organizations run public UDDI 2.0 nodes. Chapter 12 discusses the pros and cons of moving to a beta version of UDDI 2.0. Windows .NET Server Beta 3 and later supports private UDDI 2.0 directories; Microsoft no longer offers a private UDDI 1.0 implementation.

UDDI public registration of XML Web services is optional and, depending on the nature of the services you offer, might not be advisable. For example, if your XML Web services provide access to your organization’s inventory data and enable automated order entry, your marketing department undoubtedly knows the potential consumers of the services. Advertising the service in a public UDDI registry is likely to invite more hackers and competitors than prospective customers to take advantage of your commercial XML Web services. On the other hand, if you offer a general-purpose XML Web service, such as a sales tax rate lookup application, you might benefit by adding an entry for the service in a public UDDI registry.
Chapter 1: Introducing XML Web Services and the .NET Framework

W3C Security Standards: XML-Signature, XML Encryption, and XML Key Management Specification (XKMS)

HTTPS is currently the only universal (and thus currently practical) method for XML Web service consumer authentication and securing the communication channel for XML Web services that deliver confidential information. For an overview of XML Web service authentication and authorization issues in the Visual Basic 6.0 environment, read the “Building Secure Web Services with Microsoft SOAP Toolkit 2.0” paper at http://msdn.microsoft.com/library/en-us/dnsoap/html/soapsecurity.asp. Most of the recommendations in this paper also apply to ASP.NET Web Service projects.

One of the primary concerns of potential publishers of XML Web services is assuring the security of the services and their underlying data sources. Thus, W3C working groups and member organizations are busy preparing recommendations and position papers on XML-related security technology. Following are two specifications and one proposal for enhancing the security of XML Web services beyond what is offered by HTTPS:

► **XML-Signature Syntax and Processing** is a W3C recommendation, which was adopted on February 12, 2002. XML-Signature describes methods for attaching digital signatures to individual elements of an XML document or the entire document. PKI-based digital signatures are intended to verify the identity of the source of the data, such as a person authorizing a debit to a checking account. In this case, signing the SOAP request message and verifying the signature of the bank in the response message provides two-way verification.

► **XML Encryption Syntax and Processing**, as of March 4, 2002, is a W3C candidate recommendation for, according to the abstract, “a process for encrypting data and representing the result in XML. The data may be arbitrary data (including an XML document), an XML element, or XML element content. The result of encrypting data is an XML Encryption element which contains or references the cipher data.” The candidate recommendation is at http://www.w3.org/TR/xmlenc-core/.

► **XML Key Management Specification** (XKMS) is W3C note-authored by VeriSign and others for use of XML-Signature to identify cryptographic keys and authenticate request and response messages, and for use of XML Encryption to protect the confidentiality of message payload. XKMS is specifically intended to be “layered over SOAP and WSDL.” You can read the note at http://www.w3.org/TR/xkms/ and search the http://www.verisign.com with xkms to find links to Verisign’s marketing propaganda for XKMS.
Microsoft Global XML Web Service Architecture Extensions

Microsoft announced in October 2001 a set of four proposed extensions to SOAP, which have come to be known as “SOAP bubbles.” These extensions add support for SOAP-based security and message routing. Microsoft and IBM released an additional XML Web Service extension, WS-Inspection, on November 2, 2001. The five extensions were the first members of what Microsoft calls Global XML Web Services Architecture (GXA). For a long-winded description of the objectives of GXA, go to http://gotdotnet.com/team/xmlwebservices/gxa_overview.aspx.

Microsoft GXA Extensions

Following are brief descriptions of the purposes of the four original Microsoft GXA extension proposals:

► **WS-Security** Defines headers that convey the identity of XML Web service consumers and specify the methods that are used to assure confidentiality and integrity for SOAP messages.

► **WS-License** Defines headers that identify the consumer’s credentials (licenses) to use an XML Web service. WS-License is incorporated in the new WS-Security specification described in the next section.

► **WS-Routing** Defines headers for indicating the path by which SOAP messages are to be routed from one SOAP node to another.

► **WS-Referral** Defines headers that allow an individual SOAP node to specify the route to the next node, rather than including the routing information in the SOAP message header data.


The IBM/Microsoft/VeriSign WS-Security Specification

IBM, Microsoft, and VeriSign announced on April 11, 2002, a new version of the WS-Security specification that combine basic elements of the original WS-Security and WS-Licensing specs. The objective of WS-Security is to provide end-to-end message integrity, confidentiality, and non-repudiation. (HTTPS provides point-to-point security but doesn’t provide non-repudiation, which requires a digital signature for the SOAP message payload.) WS-Security specifies a set of custom SOAP headers to implement the W3C XML Encryption recommendation and XML Signature proposed recommendation. The XML Key Management Specification (XKMS) 2.0—a working draft when this book was written—will play an important role in implementing WS-Security.

IBM and Microsoft also announced a “Web services security roadmap” that includes the following six future specifications:

► **WS-Policy** for defining constraints applied by Web service security policies
WS-Trust for establishing direct and brokered trust relationships
WS-Privacy for stating and implementing Web service privacy practices
WS-Secure Conversation for managing security context and deriving session keys
WS-Federation for brokering trust relationships in heterogeneous service environments
WS-Authorization for defining and management of authorization data and policies


WS-Inspection
The WS-Inspection specification is the result of a joint effort between Microsoft and IBM to develop a Web Service Inspection Language (WSIL). The objective of WSIL, according to the authors, is to “facilitate the aggregation of references to different types of service description documents, and then provides a well-defined pattern of usage for instances of this grammar.” Unlike the GXA members of the preceding sections, WS-Inspection doesn’t involve headers.

WSIL is the replacement for Microsoft’s proprietary disco(very) service, which was the standard inspection method for .NET XML Web services prior to the Visual Studio .NET release version. You can read the full text of the proposed WS-Inspection specification at http://msdn.microsoft.com/library/en-us/dnsrvspec/html/ws-inspection.asp.

Other Proposed XML Web Service Languages
Web services tool vendors appear to be intent on erecting a modern-day Tower of WS-Babel. Proposals for new XML grammars to support real or imagined needs of XML Web services developers surface every week or two. Microsoft and IBM cooperate on many XML fronts, but the interests of IBM and its other partners often diverge from Microsoft’s. Following are three IBM-sponsored proposals that might gain widespread support:


Web Services Endpoint Language (WSEL) defines properties of endpoints, such as QoS, execution time limits, and escalation methods in the event of endpoint failure. The WSFL specification alludes to, but doesn’t define, a future WSEL implementation. WS-Endpoint might be a candidate for a future GXA extension.
Web Services Component Model (WSCM) is a project intended to standardize the browser interface for XML Web services that are directed to end users rather than servers. WSCM is based in part on Epicentric’s Web Service User Interface (WSUI) for corporate portals (http://www.epicentric.com). IBM will contribute its Web Service Experience Language (WSXL) to the project (http://www-106.ibm.com/developerworks/library/ws-wsxl/index.html). The Organization for the Advancement of Structured Information Standards (OASIS) will handle the standardization process (http://www.oasis-open.org/news/oasis_news_10_22_01.shtml).

OASIS’ primary claim to XML fame is sponsorship—in conjunction with the United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT)—of the electronic business XML (ebXML) standards. ebXML is a very complex messaging standard that’s built on top of SOAP. Microsoft arch-enemy Sun Microsystems is one of the outspoken proponents of ebXML, and the majority of ebXML implementations are Java-based.

Sun joined the general-purpose XML Web services bandwagon well after Microsoft and, when this book was written, was playing catch-up in the SOAP public relations space. Sun vice-president Rich Green said in late October 2001, “As standards are defined for the next generation of networked services—what we call services on demand—Sun ONE will have a comprehensive set of industry standards-based capabilities to offer” (emphasis added; http://www.sun.com/smi/Press/sunflash/2001-10/sunflash.20011023.5.html). When this book was written, Sun didn’t appear to be convinced that SOAP 1.1 and WSDL 1.1 were de facto industry standards because of their Microsoft/IBM heritage. Perhaps Sun plans to wait for the W3C XML Protocol Working Group to issue a formal recommendation for SOAP 1.2 before offering their “services on demand.”

XML Web Services Support in .NET
Microsoft’s .NET Framework and Visual Studio .NET represent a major departure from the company’s traditional dependence on proprietary technologies, such as COM and ActiveX, for interobject communication in the Windows environment. Standards-based XML is the lingua franca of .NET, and XML Web services are destined to be the plumbing for loosely coupled distributed applications that communicate with HTTP. .NET Remoting is the DCOM replacement for tightly coupled componentized applications. The .NET Framework, ASP.NET, and Visual Basic .NET combine to ease the transition from “legacy” COM-based components to XML Web services written in managed Visual Basic code and deployed by simple XCOPY methods.

System.Web.Services Namespaces
The .NET Framework Class Library treats everything as an object type. Even elementary datatypes are structures defined within the System object namespace. For example, the System.Int16 structure represents Visual Basic 6.0’s Integer datatype, which corresponds to Short in Visual Basic .NET.
The .NET Framework implements XML Web service elements, specifically SOAP messages and WSDL files, as classes. Transforming a class instance to its corresponding XML document or stream is called XML serialization; the reverse process is called XML deserialization. Following are brief descriptions of the primary namespaces you use when creating XML Web services and their clients:

- **System.Web.Services** contains the classes for objects that participate in the creation and consumption of XML Web services. The System.Web.Services.WebMethodAttribute class added to a Public method enables the method to be called by remote clients. The System.Web.Services.WebService class provides access to the ASP.NET Application object and HttpSessionState instance of the Session object.

- **System.Web.Services.Description** is the object representation of the WSDL file for a service. This class has 65 subclasses, which correspond to elements and attributes of WSDL documents. The System.Web.Services.Description.Mime... classes implement the SOAP Messages with Attachment specification.

- **System.Web.Services.Discovery** contains subclasses for implementing Microsoft’s DISCO machine-level service discovery mechanism, not UDDI or WS-Inspection. This class became obsolete with the release of Visual Studio .NET.


- **System.Xml.Serialization.XmlSerializer** controls serialization and deserialization of XML documents and streams. System.Xml.Serialization.Soap... classes let you customize serialization of SOAP element attributes.

Chapter 7 provides a detailed view of the System.Web.Services and System.Xml classes applicable to XML Web services.

**ASP.NET Web Service Projects**

Conventional XML Web services rely on HTTP as their transport, so the services you create with .NET run from IS 5.0+ virtual directories. ASP.NET is Visual Studio .NET’s framework for writing Web-based applications with managed code. ASP.NET automates the basic steps required to create XML Web services that are deployed as an assembly ofWebServiceName.asmx and supporting files.
TIP

Make sure the IP address of your local Default Web Site is set to (All Unassigned), not a fixed IP address, and IIS 5.0+ is running before you attempt to create your first ASP.NET Web service. The sample Web service uses localhost as the server name; if the site is bound to a fixed IP address, you receive “not found” errors when you attempt to create or run the Web service.

When you open a new ASP.NET Web Service project using Visual Basic .NET, a WebServiceName.asmx.vb file contains the code behind the .asmx file. Figure 1-13 shows the default Public Class Servicel with the <WebService()> attribute applied to define the project as a Web service, but not an XML Web service that meets this book’s definition. The http://oakleaf.ws/testws/ namespace URI replaces the default http://tempuri.org/ URI. The <WebMethod()> prefix to the Public Function HelloWorld() declaration makes the method visible to the WSDL document.

TIP

“tempuri” isn’t the plural form of “tempura.” It’s an abbreviation for “temporary Universal Resource Identifier” (URI). Replace tempuri.org with your domain name and, optionally, a virtual directory name to assure that your XML Web services have their own globally unique namespace. The URI you choose need not be accessible from the Internet; you can also use Uniform Resource Names (URNs) such as [urn:][oakleaf-ws:testws]. If you compile and run the project with tempuri.org as the URI, you receive a message that suggests changing the name.

The WebServiceName.asmx file contains only the following single line of compiler directives:

```
<%@ WebService Language="vb" Codebehind="Servicel.asmx.vb" _
Class="Servicel.Servicel" %>
```

The @WebService directive tags the file as supplying a Web service; Language="vb" specifies Visual Basic .NET as the language for the service; Codebehind designates the file containing the Visual Basic code, and the Class entry provides the equivalent of a COM object’s ProgID.

ASP.NET also adds a standard set of .NET Framework references to the project and generates a host of other supporting files in the folder linked to the IIS virtual directory, most of which appear as nodes in the Solution Explorer window. C:\DefaultSite\WebServicel is the default folder for the assembly. Double-clicking a file node displays the contents of the file in a help page of the main tabbed window.

When you uncomment the function code block and press F5 to compile and run the Web service, a test page opens in IE 5.0+ with a Service Description link to the autogenerated WSDL file. (Scan the WSDL file for the <portType>, <binding>, and <service> elements for the three invocation methods.) The HelloWorld link opens a page that lets you test the generic HTTP GET version of the service. (See Figure 1-14.) Clicking the Invoke button displays “Hello World” on an IE page. The test page also includes XML and HTML code examples for SOAP, HTTP GET, and HTTP POST request/response operations.

This brief introduction to ASP.NET’s implementation of Web services in general and XML Web services in particular offers only a very limited preview of ASP.NET’s
Chapter 1: Introducing XML Web Services and the .NET Framework

Capabilities, and is typical of trivial sample Web services that litter UDDI and private Web service registries. This book concentrates on creating and testing production-class XML Web services that deliver meaningful information and perform useful operations on SQL Server database back ends. Chapter 8, for example, shows you how to write production-grade, data-intensive B2B and B2C XML Web services with Visual Basic .NET methods and ADO.NET.

ASP.NET Pages and COM-Based XML Web Services

It's often not worth the time and effort to upgrade production Visual Basic 6.0 SOAP-enabled COM components to Visual Basic .NET. The CFR-ASPX version of the CFR application demonstrates that you can use ASP.NET pages with managed Visual Basic .NET code behind them as clients for SOAP-enabled Visual Basic 6.0 COM components. You can use the conventional ASP-style, late-bound CreateObject("MSSOAP.SoapClient") code shown in Figure 1-15 and take advantage of the .NET Framework's COM Interop feature to return an XHTML page of full-text search hits from the CFRSearchWS service.
One of Visual Studio .NET’s most useful features is the capability to early-bind XML Web services, regardless of their location, the tool used to create them, or the operating system that hosts them. Choosing Project | Add Web Reference opens a dialog of the same name with links to the Microsoft production and test UDDI registries. Typing the IP address of a .wsdl, .asmx, .disco, or .vsdisco file in the Address text box lists the available service(s) as View Contract links in the right pane and displays the contents of the WSDL file you select in the left pane. (See Figure 1-16.) Clicking the Add Reference button adds a Web References node to the Solution Explorer list. The default name of the reference you add is WebReference1; you can rename the reference to something more descriptive.

---

**Figure 1-14** This IE 6.0 test page for the HTTP GET service implementation includes samples of SOAP, HTTP GET, and HTTP POST operations.

---

TIP

You can add a remote Web Reference to a sample project by typing http://www.oakleaf.ws/cfr/CFRSearchWS.wsdl or http://66.123.163.243/cfr/CFRSearchWS.wsdl in the Add Web Reference’s Address text box.
Adding the Web Reference to your Web or Windows form project adds a Web References folder to the assembly with a subfolder for each reference. The subfolder contains a local copy of the service’s WSDL file, a Reference.map file for the service location, and a Reference.vb file, which contains the code for synchronous and asynchronous SOAP client proxies. The local copy of the WSDL file eliminates a round trip to retrieve its contents from the source site. Early-binding the Web Reference reduces the number of lines of code to retrieve the SOAP message payload from six to two. (See Figure 1-17.) A simple function call passes the required arguments to the SOAP request message and delivers the response message payload to an ASP.NET Web page. You also gain the benefits of statement completion with autolist members and parameter data.

![Figure 1-15](image)

Figure 1-15 This Visual Basic .NET code behind the Search.aspx ASP.NET page late-binds the CFRSearchWS XML Web service with a COM-based MSSOAP.SoapClient proxy.

Adding the Web Reference to your Web or Windows form project adds a Web References folder to the assembly with a subfolder for each reference. The subfolder contains a local copy of the service’s WSDL file, a Reference.map file for the service location, and a Reference.vb file, which contains the code for synchronous and asynchronous SOAP client proxies. The local copy of the WSDL file eliminates a round trip to retrieve its contents from the source site. Early-binding the Web Reference reduces the number of lines of code to retrieve the SOAP message payload from six to two. (See Figure 1-17.) A simple function call passes the required arguments to the SOAP request message and delivers the response message payload to an ASP.NET Web page. You also gain the benefits of statement completion with autolist members and parameter data.

**NOTE**

An interesting feature of ActiveX DLLs with SOAP wrappers applied by the SOAP Toolkit 2.0 is that the underlying object isn’t affected. You can invoke the object by a conventional method call or a SOAP client proxy. The CFR project’s CFR-COM, CFR-SOAP, and CFR-ASPX implementations share a single set of Visual Basic 6.0 ActiveX DLLs.
Your “take-away” from these few brief sections that describe .NET support for XML Web services, as well as much of the chapter’s earlier content, should be this:

Visual Studio .NET’s developers have made use of XML Web services transparent to Visual Basic developers. Invoking an XML Web service written in Java, COBOL, C++, C#, or Visual Basic that’s accessible from any Web server anywhere on the Internet is as easy as instantiating a local COM or Visual Basic .NET component.

Following are the obligatory action steps:

► Consider SOAP-enabling all your existing ActiveX DLLs that deliver data as XML documents.
► Write all new Visual Basic .NET components as ASP.NET [XML] Web Services, unless you have a good reason to do otherwise.
Figure 1-17  Early-binding XML Web services makes their invocation as simple as that for traditional early-bound COM objects.

- Decide which SQL Server stored procedures to enable for direct XML Web service access.
- Take advantage of early-bound XML Web services in all new .NET Web and Windows form clients.
- Read the rest of this book and try the example code to prepare for the “next big thing” in distributed computing.