
Module 6: Using Web Services

Contents

Overview	1
What Is a Web Service?	2
Calling a Web Service from a Browser	12
Calling a Web Service by Using a Proxy	15
Creating a Simple Web Service by Using Visual Basic	22
Creating and Calling a Web Service by Using Visual Studio .NET	26
Lab 6: Using Web Services	35
Review	47



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, places or events is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2001 Microsoft Corporation. All rights reserved.

Microsoft, MS-DOS, Windows, Windows NT, ActiveX, FrontPage, IntelliSense, Jscript, Outlook, PowerPoint, Visual Basic, Visual InterDev, Visual C++, Visual C#, Visual Studio, and Windows Media are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Overview

- What Is a Web Service?
- Calling a Web Service from a Browser
- Calling a Web Service by Using a Proxy
- Creating a Simple Web Service by Using Visual Basic
- Creating and Calling a Web Service by Using Visual Studio .NET

The Web has helped to facilitate better communication within and between companies by providing fast access to information. For many organizations, however, browsing data-driven pages does not adequately satisfy their business needs. Programmable Web sites that directly link organizations, applications, services, and devices with one another would better meet these needs.

Web services provide a simple, flexible, standards-based model for binding applications together over the Internet, and taking advantage of existing infrastructure and applications. Web applications can easily be assembled by using locally developed services and existing services, regardless of which platforms, development languages, or object models have been used to implement the constituent services or applications.

After completing this module, you will be able to:

- Explain the goal of Web services and how they fit into the Web architecture.
- Describe the Web services execution model.
- Call a Web service from a browser.
- Call a Web service by using a proxy.
- Use the data returned by a Web service.
- Create a simple Web service by using Microsoft® Visual Basic®.

◆ What Is a Web Service?

- Features of Web Services
- Web Services Execution Model
- Finding Existing Web Services
- Examples of Existing Web Services
- Demonstration: Calling a Web Service from the Browser

Web services provide a simple, flexible, standards-based model for binding applications together over the Internet and taking advantage of the existing infrastructure and applications. In this section, you will learn about the need for Web services and about their architecture. You will also learn about existing Web services and how to find them.

Features of Web Services

- Allow Applications to Communicate Across the Internet
- Language Independent
- Protocol Independent
- Platform Independent
- Stateless Architecture

One of today's most pressing challenges for developers is application integration. Application integration is the process of taking a group of applications and turning them into easy-to-use Web applications, even if they run on different operating systems, were created with different programming languages, and were built with different object models.

Like components, Web services represent black-box functionality that developers can reuse without worrying about how to implement the service. Web services provide well-defined interfaces (called contracts) that describe the services they represent.

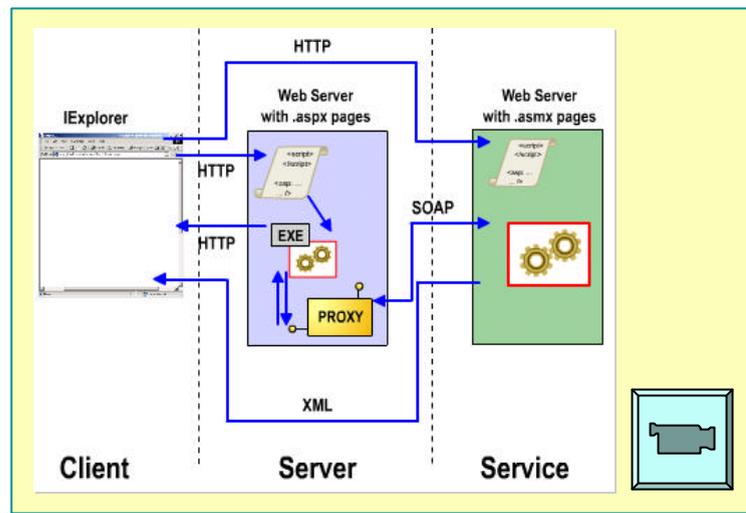
Developers can assemble applications by using a combination of remote services, local services, and custom code. For example, a company might assemble an online store that uses the Microsoft Passport service to authenticate users, a third-party personalization service to adapt Web pages to each user's preferences, a credit-card processing service, a sales tax service, package-tracking services from each shipping company, an in-house catalog service that connects to the company's internal inventory management applications, and custom code to individualize the interface and make it unique.

Unlike current component technologies, however, Web services do not use protocols that are specific to certain object models, such as the Distributed Component Object Model (DCOM), which requires specific, homogeneous infrastructures on the computers that run the client and the server. Web services communicate by using standard Web protocols and data formats, such as Hypertext Transfer Protocol (HTTP), Extensible Markup Language (XML), and Simple Object Access Protocol (SOAP). Any system that supports these Web standards can support Web services.

A Web service can be used internally by a single application, or it can be used externally by many applications that access it through the Internet. Because it is accessible through a standard interface, a Web service allows disparate systems to work together. The Web services model is independent of languages, platforms, and object models.

The Web service model is supported by Microsoft ASP.NET, which is a unified Web development platform that has grown from Active Server Pages (ASP) technology. The ASP.NET Web services model assumes a stateless service architecture. Stateless architectures are generally more scalable than stateful architectures. Each time a service request is received, a new object is created. There is a request for the method call, the method call is returned, and then the object is destroyed. Services can use the ASP.NET State Management services to maintain a state between requests.

Web Services Execution Model



The Web services execution model involves two entities: the client and the service provider.

Note The term “client” is often misinterpreted. In this sense, a client is a Web browser that views the results of calling a Web service.

The preceding slide illustration shows how Web services are used between a client and the Web server. Each component has a specific role in the execution model.

Web Service Model

In the Web service model, the Web service developer:

1. Creates the .asmx file that includes the namespace, classes, properties, and methods.
2. Declares methods as Web methods that can be accessed over the Internet.

The following is an example of a simple .asmx file :

```
<%@ WebService Language="VB" Class="MathService" %>
```

```
Imports System.Web.Services
```

```
Imports System
```

```
Class MathService
```

```
<WebMethod()> Public Function Add(int1 As Integer, _
    int2 As Integer) As Integer
    return(int1 + int2)
```

```
End Function
```

```
End Class
```

Client

In the Web service model, the client:

1. Calls the Web service from the browser to determine which methods are available.

When you call a Web service from a browser, you access the description page, which lists the methods that are included in the Web service. The protocol that is used in this case is HTTP, and the data is returned as XML.

2. Calls a method of the Web service from the browser.

When you call a method of a Web service from a browser, the protocol that is used is HTTP, and the data is returned as XML.

Web Server

You can also call methods of the Web service by using code on an ASP.NET page. To call a Web service from an ASP.NET page, the developer must:

1. Find out which Web services are available. This involves finding the Uniform Resource Locator (URL) for the Web service.
2. Compile the .asmx file (Web service) into a proxy by using the Wsd.exe utility. This creates a .vb or .cs file that contains source code for the proxy.

You will learn about proxy files and how to compile an .asmx file into a proxy in the next section of this module.

3. Compile the proxy (the .vb or .cs file you created in the previous step) into a .dll file in the **/bin** directory of the Web site.
4. Open the ASP.NET Web page.
 - a. Create an instance of the proxy.
 - b. Call the methods of the Web service.
 - c. Use the data returned by the Web service.

Note As the preceding slide illustration shows, the proxy resides on the same server as the Web page (.aspx) that calls it. An advantage of using a proxy is that it translates the returned data from the server into a data type that the .aspx page can use. If you call a Web service method directly from an .aspx page without using a proxy, you will receive the data in XML form, which must be parsed in another step.



To learn more about the Web services execution model, view the **Web Services Execution Model** animation. Open the file **2063B_06A001.swf** from the **Media** folder.

Finding Existing Web Services

- **UDDI**

- Defines a way to publish and discover information about Web services
- Relies on a distributed registry of businesses and their service descriptions implemented in a common XML format

- **Businesses Individually Register Information About the Web Services by Using UDDI Business Registration**

<http://www.uddi.org>

<http://uddi.microsoft.com>

Getting information about which business partners have Web services and finding specific information about the available Web services is very difficult for companies today. The Universal Description, Discovery and Integration (UDDI) specification defines a way to publish and discover information about Web services.

UDDI relies on a distributed registry that has been implemented in a common XML format. This registry is a list of businesses, along with descriptions of their available services. The UDDI specification consists of several related documents and an XML schema that defines a SOAP-based programming protocol for registering and discovering Web services.

The core component of UDDI is the UDDI business registration, an XML file that is used to describe a business entity and its Web services. Conceptually, the information provided in a UDDI business registration consists of three components: White Pages, which include address, contact, and known identifiers; Yellow Pages, which include industrial categorizations based on standard taxonomies; and Green Pages, which include the technical information about the services that are exposed by the organization. Green Pages include references to specifications for Web services, as well as any support that may be required for pointers to file and URL-based discovery mechanisms.

Businesses individually register information about the Web services that they expose for other businesses to use. This information can be added to the UDDI business registry through a Web site, or by using tools that use the programmatic service interfaces described in the UDDI programmer's application programming interface (API) specification. The UDDI business registry is a logically centralized, physically distributed service with multiple root nodes that replicate data with each other regularly. When a business registers with a single instance of the business registry service, the service shares the data automatically with other UDDI root nodes. After the data has been distributed, it becomes freely available to anyone who needs to discover which Web services are exposed by a particular business.

For more information about UDDI, go to the UDDI Web site at <http://www.uddi.org/> or the Microsoft UDDI Project Web site at <http://uddi.microsoft.com>

Examples of Existing Web Services

- **Find Web Services at:**
 - <http://www.xmethods.net>
 - <http://www.gotdotnet.com>
 - <http://dotnet.microsoft.com>

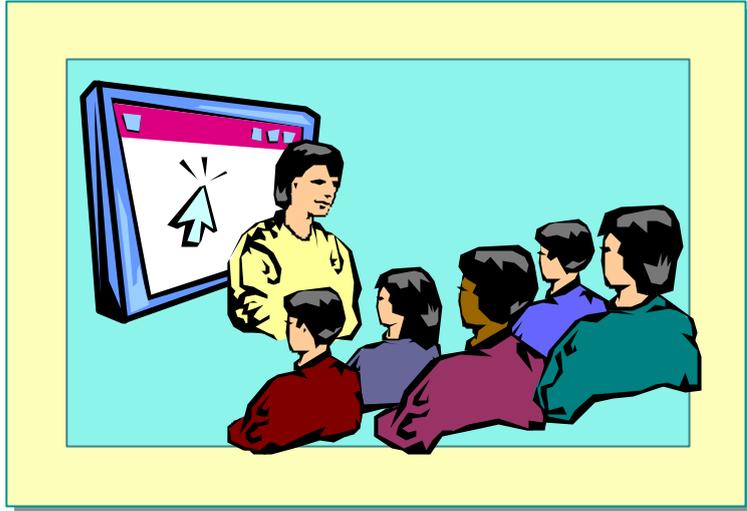
You can find existing Web services at the following Web sites:

- <http://www.xmethods.net>
- <http://www.gotdotnet.com>
- <http://dotnet.microsoft.com>

Some examples of Web services that can be found at these sites include:

- **FedEx Tracker**
Provides access to FedEx Tracking information.
- **Weather-temperature**
Gives the current temperature in a given United States zip code region.
- **Barnes and Noble Price Quote**
Returns the price of a book at barnesandnoble.com when you provide an International Standard Book Number (ISBN).

Demonstration: Calling a Web Service from the Browser

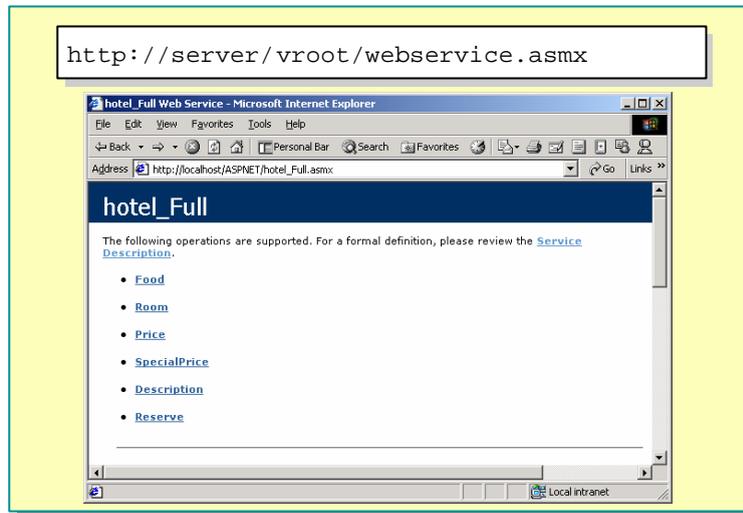


In this demonstration, you will see how to call a Web Service.

↳ To run the demonstration

1. View the description page for the MathService Web service at <http://localhost/2063/Democode/Mod06/mathservice.asmx>
2. Click the **Add** link.
3. Fill in parameters for the **Add** method, and then click **Invoke**.
4. Return to the mathservice.asmx page and view the Web Service Description Language (WSDL) contract.

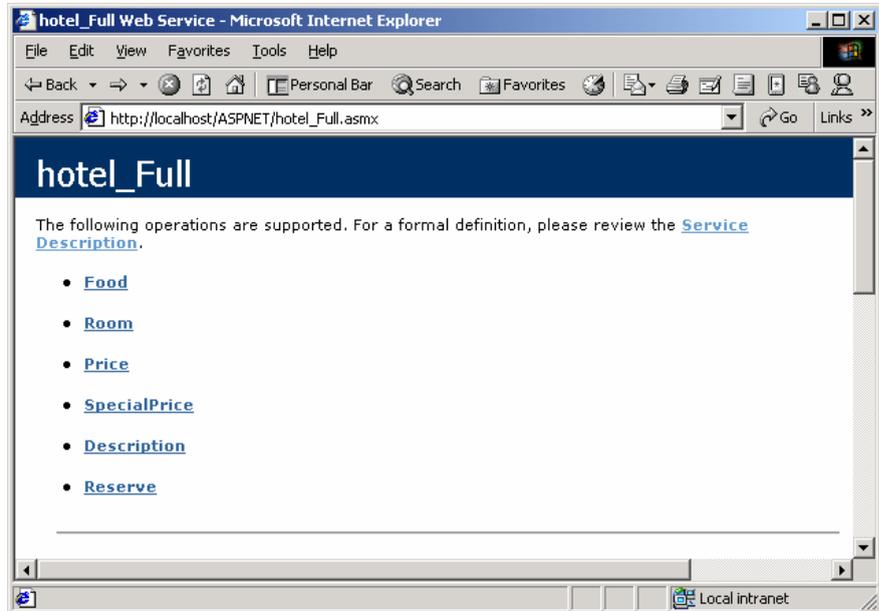
Calling a Web Service from a Browser



Because Web Services are accessible by using URLs, HTTP, and XML, programs running on any platform and in any language can access them.

If you know the base URL for a Web service (the URL to the .asmx file that is the base of the Web service), you can use this URL to access a Web page known as the HTML description page. The HTML description page provides information about what a Web service does, the methods it contains and their parameters, and its response type. In addition, you can use the description page to test the functionality of the Web service.

For example, suppose that you access a Web service called `hotel_Full.asmx` that is used to retrieve details from a hotel. You know the base URL for this service is `http://localhost/ASPNET/hotel_full.asmx`. Entering the base URL with no extensions or parameters produces a page that displays information about the service and the methods it contains.



In addition to viewing information about the Web service in the browser, you can obtain a more formal definition of the Web service by viewing its Web Service Description Language (WSDL) contract. A **Service Description** link at the top of the description page allows you to view the contract, which contains an XML description of the Web service and its contents. You can use this file to generate a proxy manually.

WSDL Contract

The WSDL contract is an XML document. The document defines the format of messages that the Web service understands. The service description acts as an agreement that defines the behavior of a Web service and instructs potential clients in how to interact with it.

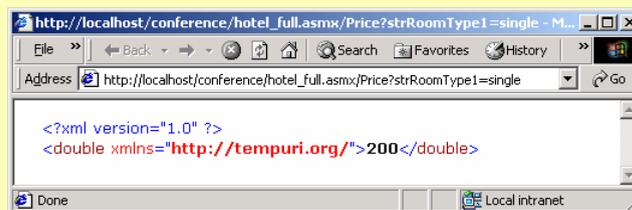
In addition to message format definitions and messaging patterns, the service description also contains the address that is associated with each Web service entry point. The format of this address depends on the protocol that is used to access the service, such as a URL for HTTP or an e-mail address for Simple Mail Transfer Protocol (SMTP).

Calling a Web Service from a Browser (*continued*)

- Pass the Name of the Method and All Required Parameters to the URL of the Web Service

```
http://server/vroot/webservice.asmx/method?param=value
```

- Return Value Is in XML Format



You can also call the methods of a Web service directly from a browser.

To call a Web service method, pass the name of the method, the required parameters, and the values of the parameters to the URL of the Web service. The Web service returns data in XML format when it is called from a Web browser by using HTTP.

For example, if a Web service named `hotel_full` has a method named **Price** that takes one parameter named **strRoomType1**, you can call it directly from the browser by viewing the following URL:

```
http://localhost/conference/hotel_full.asmx/Price?strRoomType1=single
```

The following XML is data that was returned from this URL:

```
<?xml version="1.0" ?>
<double xmlns="http://tempuri.org/">200</double>
```

If you want to use this kind of data in your Web application, you must parse it in a separate step.

◆ Calling a Web Service by Using a Proxy

- Compiling an .asmx File into a Proxy
- Compiling the Proxy into a DLL
- Calling Web Service Methods
- Demonstration: Calling a Web Service by Using a Proxy

Another method for calling a Web service is by using a proxy. In this section you will learn how to call a Web service by using a proxy.

Note You will first learn how to create a Web service manually, and then learn how to create it by using Microsoft Visual Studio® .NET.

Compiling an .asmx File into a Proxy

■ A Proxy Class:

- Looks like the original class but does not contain any application logic
- Allows a client to access a Web service as if it were a local COM object

■ Creating a Proxy Class from an .asmx File

```
wsdll /l:vb /n:myNameSpace  
      http://location/service.asmx?wsdl
```

A proxy class is code that looks exactly like the class it is meant to represent, but does not contain any of the application logic. Instead, it contains marshaling and transport logic. The **Proxy** object allows a client to access a Web service as if it were a local Component Object Model (COM) object, which lets any COM-enabled language access the Web service easily. You can create a proxy class from an SDL file.

The Microsoft .NET Framework software development kit (SDK) includes a tool called Wsdll.exe that you can use to create a proxy class.

Syntax

```
wsdll [options] {URL | path}
```

In the preceding syntax, **URL** is the Uniform Resource Location (URL) to a WSDL contract file (.wsdl), XSD schema file (.xsd), or a discovery document (.disco). **Path** is the path to a local WSDL contract, XSD schema, or discovery document (including .discomap files).

The following table provides a detailed description of the options that are used with the Wsd.exe command.

Option	Description
/urlkey	Specifies the configuration key to use to read the default value for the URL property when generating code. Also called <code>/appsettingurlkey</code> .
/baseurl	Specifies the base URL to use when calculating the URL fragment. You must use the <code>/urlkey</code> option with this option. Also called <code>/appsettingbaseurl</code> .
/d[omain]	Specifies the domain name to use when connecting to a server that requires authentication.
/l[anguage]	The language that is used for the generated proxy: CS (Microsoft Visual C#™; default), VB (Microsoft Visual Basic), or JS (Microsoft JScript® .NET).
/n[amespace]	The namespace of the generated proxy. Default value is the global namespace.
/nologo	Suppresses the Microsoft startup banner display.
/o[ut]	The location in which to create the proxy file and the filename for the file. The default location is the current directory; the default filename is based on the Web Service name.
/p[assword]	Specifies the password to use if connecting to a server that requires authentication.
/protocol	The protocol that is used to generate the proxy: SOAP, HttpGet, HttpPost, or a custom protocol as specified in the configuration file. Default value is SOAP.
/proxy:	The URL of the proxy server that is used to establish a connection. Default uses the system proxy settings. Associated with this are three other options: <ul style="list-style-type: none"> ■ <code>/pd</code> (or <code>/proxydomain</code>) which sets the domain in which the proxy server resides, ■ <code>/pp</code> (or <code>/proxypassword</code>) which sets the password required by the proxy server, and ■ <code>/pu</code> (or <code>/proxyusername</code>) which sets the username required to connect to the proxy server.
/server	Generates an abstract class for a Web service that is based on the contract. Default generates a client proxy class.
/u[sername]	Specifies the username to use when connecting to a server that requires authentication.
/?	Displays command syntax and options for the Wsd.exe tool.

Wsd.exe generates a single source file as output. This file contains a proxy class that has been defined by using the specified language and the methods that are exposed by the Web service. Each proxy method contains the appropriate network invocation and marshalling code that is necessary to invoke and receive a response from the remote Web service.

For example, to compile a store Web service file named Store.asmx into a proxy, you could type the following code:

```
wSDL /l:vb /n:myStore  
http://localhost/Store/Store.asmx?wsdl
```

Here, myStore is the namespace of the Web service.

This code creates a proxy class in a file named Store.vb that contains methods that are exposed by the Web service.

Compiling the Proxy into a DLL

- Compiling a Proxy Class into a DLL

```
vbc /out:..\bin\proxyclassname.dll  
/t:library  
/r:System.Web.Services.dll proxyclassname.vb
```

- The .dll File Is Published in the /bin Folder

After you create a proxy class, you need to compile it into a dynamic-link library (DLL). You can use the following syntax to compile a Visual Basic proxy class into a DLL:

```
vbc /out:..\bin\proxyclassname.dll /t:library  
/r: System Web. Servi ces. dll proxyclassname.vb
```

In the preceding code's syntax, you use the /t switch to specify the target output type of the compiled resource. You use the value **library** to create a stand-alone file that contains the class and the metadata for a given class or classes. You use the /r switch to reference an assembly that contains the metadata and classes that you need within the class instance.

Example

If you want to compile the **Store.vb** class into a DLL, you could use the following code:

```
vbc /out:..\bin\Store.dll /t:library  
/r: System Web. Servi ces. dll store.vb
```

Note that in the preceding code's syntax, the output is stored in the /bin folder. To use the proxy DLL, you need to publish the DLL in the /bin folder. The /bin folder resides in the root of the Web virtual site, or in the root in Internet Information Services (IIS).

Calling Web Service Methods

■ Creating an Instance of the Proxy

```
Dim myStoreProxy As myStore.Store  
myStoreProxy = New myStore.Store()
```

■ Calling Methods

```
Dim ds As DataSet  
ds = myStoreProxy.products()
```

Create an Instance

To use a proxy in an .aspx page, you need to create an instance of the proxy in your page. For example, to use the **Store** proxy, you create an instance of it by writing the following code:

```
Dim myStoreProxy As myStore.Store  
myStoreProxy = New myStore.Store()
```

In the preceding code, **myStore** is the namespace of the proxy class, and **Store** is the actual proxy class.

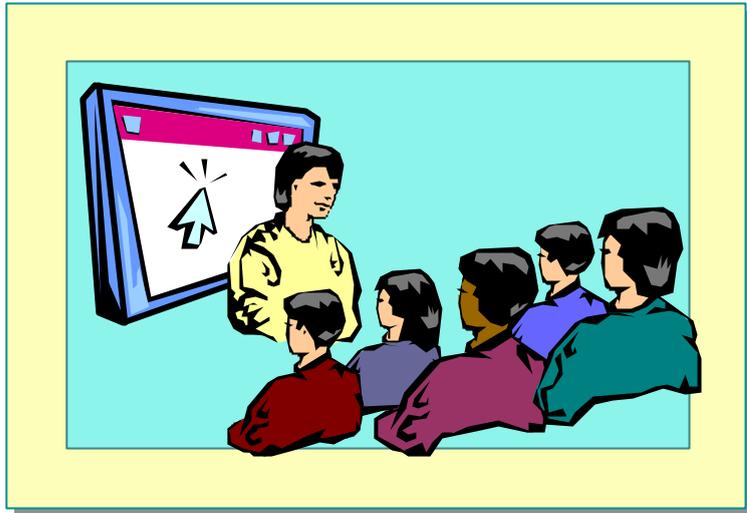
Call a Method

You can call the methods of a Web service from an .aspx page directly by using the proxy, and you can then access the data that the Web service returns.

For example, if the **Store** class has a Web-callable method named **products** that returns a **DataSet** of all the products sold by the store, you could call the method by using the following code:

```
Dim ds As DataSet  
ds = myStoreProxy.products()
```

Demonstration: Calling a Web Service by Using a Proxy



In this demonstration, you will see how to create and use a proxy to call a Web service.

⚡ To run the demonstration

1. Create a bin directory in the folder *<install folder>*.

This is the virtual root directory of the 2063 Web site.

2. Build the proxy DLL for the MathService Web service by running *<install folder>\Democode\Mod06\mkService.bat*.

This batch file creates a proxy DLL named mathserver.dll and moves it to the \bin directory.

3. Open the file *<install folder>\Democode\Mod06\math.aspx* to show how to call the Web service from code.
4. View math.aspx in Microsoft Internet Explorer.
5. Enter numbers in the two text boxes, and then click =.

◆ Creating a Simple Web Service by Using Visual Basic

- Writing a Web Service
- Demonstration: Creating a Web Service

The biggest advantage of Web services is that they can be created easily. In this section, you will learn how to create a Web service and expose methods.

Writing a Web Service

■ Adding a Page Directive

```
<%@ WebService Language="VB" Class= "MyStore" %>
```

■ Importing the Namespaces

```
Imports System  
Imports System.Web.Services
```

Writing a simple Web service takes only a few minutes, and can be accomplished in any text editor.

As an example, consider a Web service called MyStore. MyStore exposes a method for getting the price for all the products in the store.

Adding the Page Directive

At the top of the .asmx file is a directive that identifies the file as a Web service and specifies the language for the service.

```
<%@ WebService Language="VB" Class="MyStore" %>
```

Importing the Namespaces

Next, import the System and the System.Web.Services namespaces. The **WebMethod()** attribute resides in the System.Web.Services namespace. You can use the **WebMethod()** attribute to call a method from the Web.

```
Imports System  
Imports System.Web.Services
```

Writing a Web Service (*continued*)

- **Defining a Class with WebMethods**

```
Public Class MyStore
  <WebMethod()> Public Function _
    Price() As String
    Price = "The item costs $10."
  End Function
End Class
```

- **Save the File with an .asmx Extension**

Defining a Class File

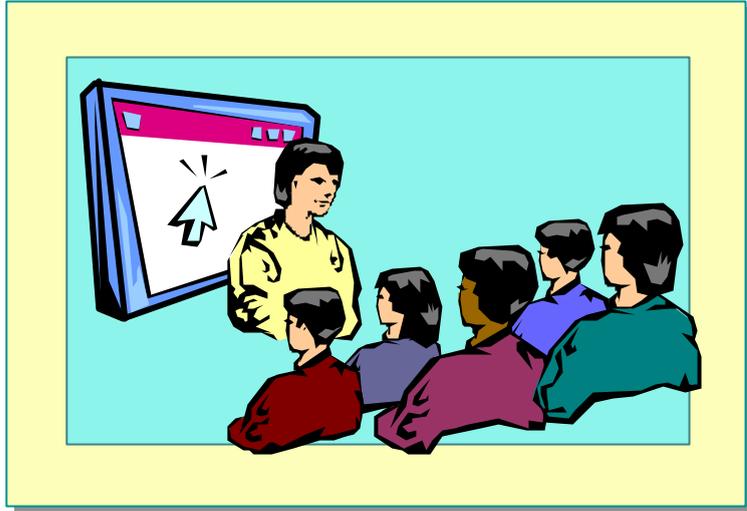
In the same .asmx file, you must also define a class that encapsulates the functionality of the Web service. This class should be public, and inherit from the `WebService` base class. Each method that will be exposed from the service needs to be flagged with a `<WebMethod()>` custom attribute in front of it. This attribute is required to create a Web-callable method. If the method does not have the `<WebMethod()>` custom attribute, the method will not be exposed from the service. The following code has the `MyStore` class and the `Price` method:

```
Public Class MyStore
  <WebMethod()> Public Function Price() As String
    Price = "The item costs $10."
  End Function
End Class
```

Saving a Web Service

Web Service files are saved with an .asmx file extension. The ASP.NET runtime automatically compiles files that have an .asmx extension when a request to the service is made.

Demonstration: Creating a Web Service



In this demonstration, you will see how to create a simple Web service that subtracts two numbers.

↳ To run the demonstration

1. Edit the file `<install folder>\Democode\Mod06\MathService.asmx`.
2. Create a **Subtract** method that subtracts one number from another.
3. View the page <http://localhost/2063/Democode/Mod06/mathservice.asmx>
4. Fill in parameters for the **Subtract** method, and then click **Invoke**.
5. If you want to call this new method service from an ASP.NET page, you can use the `mkservice.bat` file to compile it. Then call the **Subtract** method in the `math.aspx` page.

◆ Creating and Calling a Web Service by Using Visual Studio .NET

- Creating a Web Service by Using Visual Studio .NET
- Demonstration: Creating a Web Service by Using Visual Studio .NET
- Calling a Web Service by Using Visual Studio .NET
- Demonstration: Calling a Web Service by Using Visual Studio .NET

Visual Studio .NET makes creating and calling Web services easy.

In Visual Studio .NET, you can easily expose any function, in any language, as a Web service. You do not need to learn XML and SOAP to take advantage of Web services. When you compile your business objects, Visual Studio .NET automatically generates an XML file that describes the function and, when it is called, the function will automatically send and receive XML packets.

After you build the Web service, both the compiled code and the XML file that describes the public methods of the service are published to the Web server. The Web service can then be invoked through HTTP, and XML will automatically be used to pass data to and from the service.

In Visual Studio .NET, you can drag any exposed Web service directly into your application. Doing so enables Visual Studio .NET to treat the Web service as a class. Calling the Web service is as simple as creating a new instance of the Web Service class and then calling an exposed method.

In this section, you will learn how to create and call Web services by using Visual Studio .NET.

Creating a Web Service by Using Visual Studio .NET

- **Creating a Web Service:**
 1. Create a new project in Visual Studio .NET
 2. Select the ASP.NET Web Service option and specify the name of the Web service
 3. Open the Web service in the code view and declare its Web-callable functions
 4. Compile the Web service by using the Build option

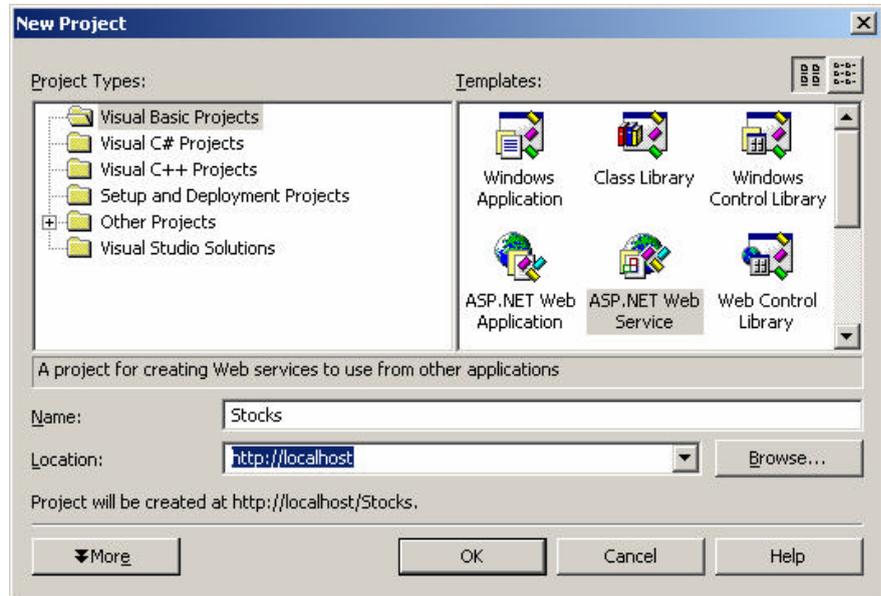
When you use Visual Studio .NET to create a Web service, you do not need to add several lines of code that you need to add when using Microsoft Notepad or another text editor.

You can simply declare the Web-callable functions of the Web service. Visual Studio .NET automatically creates the code for adding pages directives, importing namespaces, and declaring the class structure when you create a new project for the Web service.

To learn how to create a Web service by using Visual Studio .NET, consider a simple Web service called Stocks.

⏪ **To create a Web service in Visual Basic**

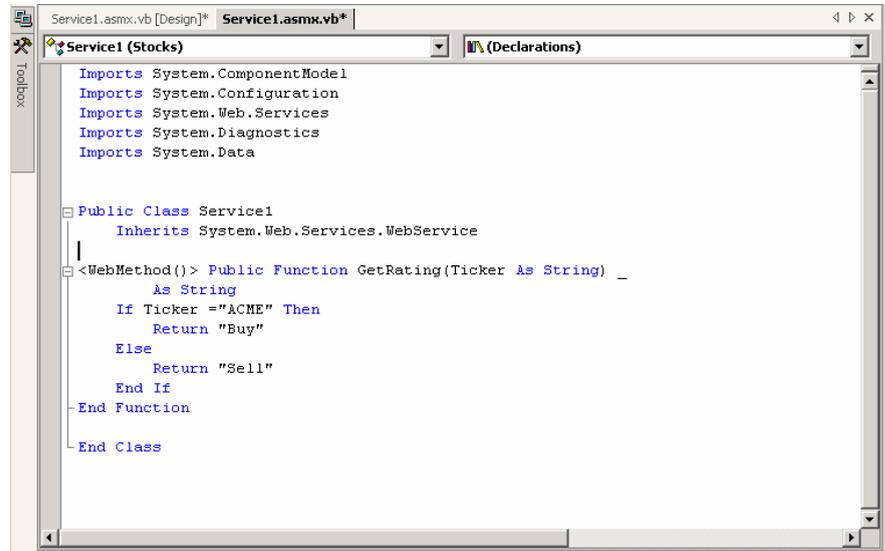
1. Create a new project in Visual Studio .NET.
2. Select the **ASP.NET Web Service** option, type a name for the Web service, and then click **OK**.



By default, Visual Studio .NET creates a Web service file named Service1.asmx.

Note You can change the default name of the Web service file (Service1.asmx) and the default name of the class (Service1).

3. Open the .asmx file in code view, and then declare its Web-callable functions. For example, the following illustration shows a **GetRating** method.



```
Service1.asmx.vb [Design]* Service1.asmx.vb*
Service1 (Stocks) (Declarations)
Imports System.ComponentModel
Imports System.Configuration
Imports System.Web.Services
Imports System.Diagnostics
Imports System.Data

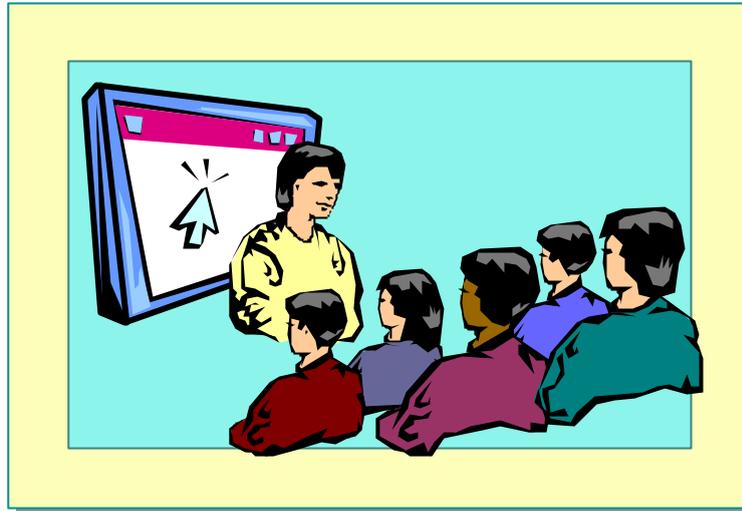
Public Class Service1
    Inherits System.Web.Services.WebService

    <WebMethod()> Public Function GetRating(Ticker As String) _
        As String
        If Ticker = "ACME" Then
            Return "Buy"
        Else
            Return "Sell"
        End If
    End Function
End Class
```

4. Compile the Web service by clicking **Build** on the **Build** menu.

Visual Studio .NET compiles the Web service code and deploys it as a DLL to the \bin directory of the Web site.

Demonstration: Creating a Web Service by Using Visual Studio .NET



In this demonstration, you will learn how to create a simple stock rating service. The rating service returns a Buy value if the ticker symbol ACME is entered.

↳ To run this demonstration

1. Create a new project in Visual Studio .NET.

It is important to create a new project for the ASP.NET Web service.

2. Select the **ASP.NET Web Service** option, and then specify the name of the Web service as Stocks.

A Stocks folder is created in the folder \Inetpub\wwwroot.

3. Open the default .asmx file, Service1.asmx, in code view by right-clicking the file in the Solution Explorer and clicking **View Code**.

The default name of the class is Service1 (the code is in the file Service1.asmx.vb).

4. Create a **GetRating** function as follows:

```
<WebMethod()> Public Function GetRating(Ticker As String) _
    As String
    If Ticker ="ACME" Then
        Return "Buy"
    Else
        Return "Sell"
    End If
End Function
```

5. Save the file.
6. On the **Build** menu, click **Build** to compile the Web service.
7. Open the Web service in Internet Explorer by viewing <http://localhost/Stocks/Service1.asmx>

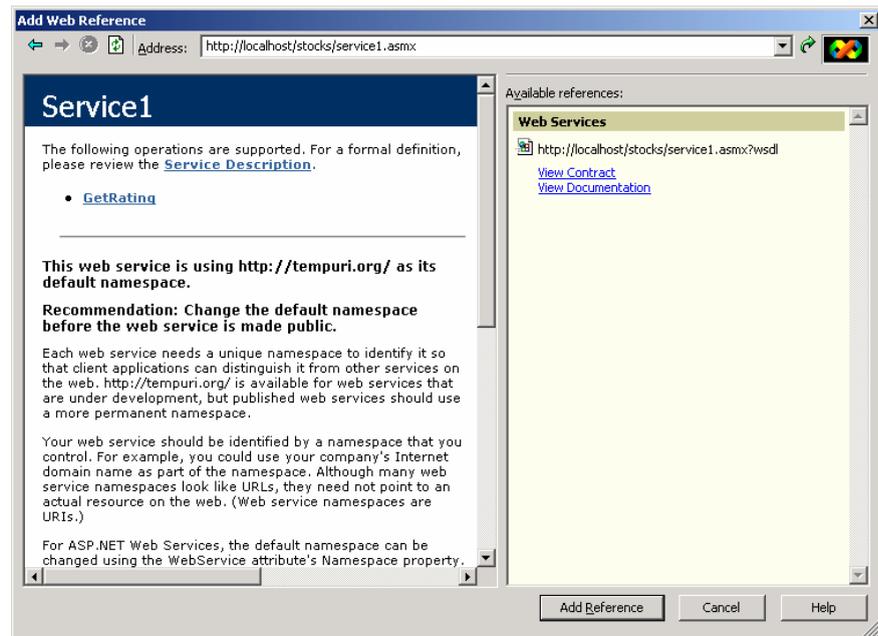
Calling a Web Service by Using Visual Studio .NET

- **Calling a Web Service:**
 1. Open a Web Application project
 2. Create a Web reference for the Web service by using the Add Web Reference dialog box
 3. Create an instance of the Web service
 4. Call the functions of the Web service
 5. Build the Web application project

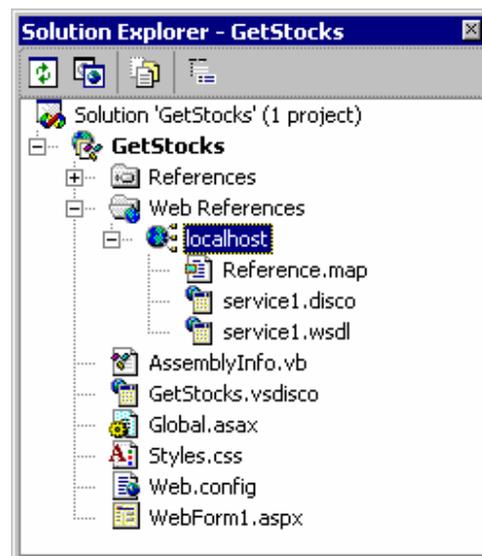
To use a Web service from Visual Studio .NET, you can simply add a Web reference to it. After browsing to the Web service URL, you can reference the service from your project. Because the WSDL for the Web service contains both the URL of the Web service and all of the functions that are available, Visual Studio .NET can automatically create the code that calls the service from your page.

☞ **To call a Web service by using Visual Studio .NET**

1. Open a Web Application project in Visual Studio .NET.
2. On the **Project** menu, click **Add Web Reference**.
3. In the **Address** field of the **Add Web Reference** dialog box, type the URL of the Web service that you are accessing, press ENTER, and then click **Add Reference**.



Visual Studio .NET creates a Web reference to the Web service with the name of the server hosting the Web service, as shown in the following illustration.



4. In a function in an ASPX page, create an instance of the Web service, and then call its functions.

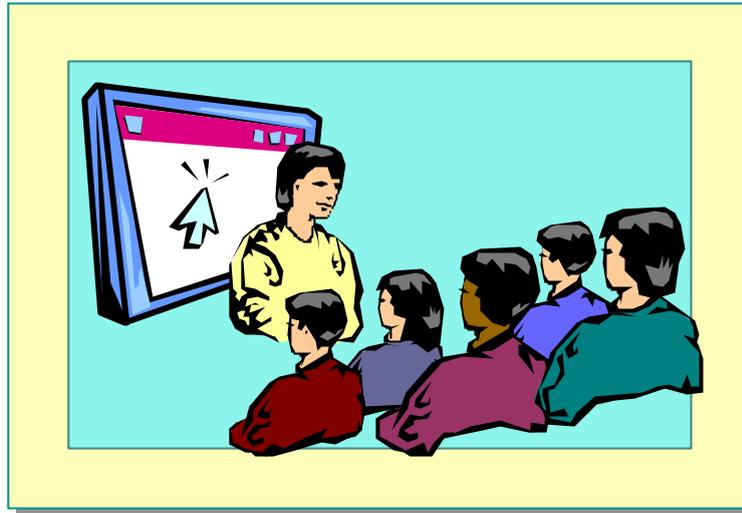
For example, if you have a button to call the **GetRating()** method of the Stocks Web service, use the following code in the **Click** event procedure:

```
Sub Button1_Click(s As Object, e As EventArgs)
    Dim service As New GetStocks.local host. service1()
    lblResults.Text = service.GetRating("ACME")
End Sub
```

Note When you use Web services in a Visual Studio .NET project, the proxy is built for you and added to the assembly for the project. Therefore, to reference a Web service, you must use the name of the project, the name of the Web reference, and then the name of the WSDL file created by Visual Studio .NET.

5. Compile the Web application by clicking **Build** on the **Build** menu.

Demonstration: Calling a Web Service by Using Visual Studio .NET



In this demonstration, you will learn how to call a Web service called Stocks by using Visual Studio .NET.

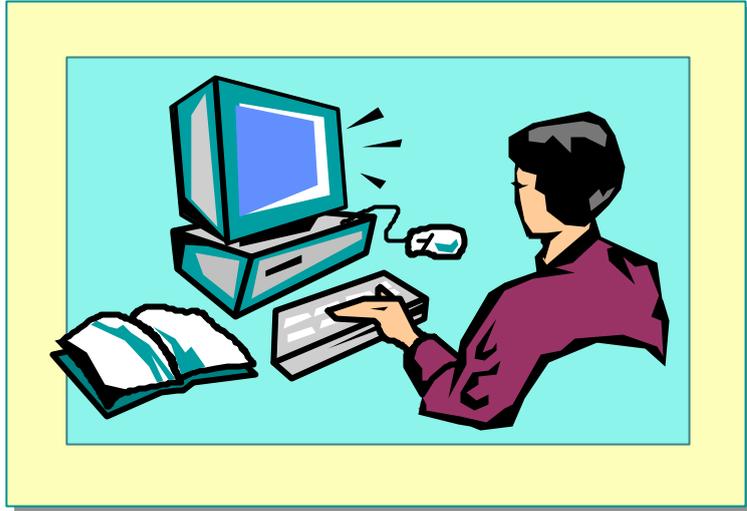
↳ To run the demonstration

1. Create a new ASP.NET Web Application project in Visual Studio .NET called **GetStocks**.
2. Add a Web reference to the Web service that you created in the previous demonstration. It should be located at <http://localhost/Stocks/service1.asmx>
3. Open the default Webform1.aspx page in design view and add a text box, a button, and a label using the Web Forms section of the Toolbox. Use the default properties for each.
4. Create a **Click** event procedure for the button.

```
Private Sub Button1_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles Button1.Click  
    Dim service As New GetStocks.Localhost.service1()  
    Label1.Text = service.GetRating(TextBox1.Text)  
End Sub
```

5. Build the Web application by clicking **Build** on the **Build** menu.
6. View the .aspx page in Internet Explorer.
7. Test the Web service by entering the ACME stock symbol in the text box and clicking the button.

Lab 6: Using Web Services



Objectives

After completing this lab, you will be able to:

- Call a Web service from an ASP.NET page.
- Create a Web service by using Visual Basic.

Prerequisites

Before working on this lab, you must know:

- How to create an event procedure for a server control.
- How to create a class in Visual Basic.

Lab Setup

There are starter and solution files associated with this lab. The starter files are in the folder *<install folder>\Labs\Lab06\Starter* and the solution files for this lab are in the folder *<install folder>\Labs\Lab06\Solution*.

Scenario

After selecting a track for the ASP.NET conference, the user can reserve a room in a hotel for the duration of the conference. From the CheckOut.aspx page, it is possible to select one of two hotels at which to stay. Selecting a hotel calls the hotel_reservation.aspx page.

There are two hotels that have contracted with the conference. For the purposes of this lab, one hotel will always be full; the other will always have rooms available.

In Exercise 1, you will add code to the hotel_reservation.aspx page to call an existing Web service to get information about the Full hotel.

In Exercise 2, you will create a Web service for a third hotel by using Visual Basic.

In Exercise 3, you will add code to the hotel_reservation.aspx page to get information about the second hotel, hotel_Empty.

Estimated time to complete this lab: 60 minutes

Exercise 1

Using a Web Service

In this exercise, you will call a Web service for a hotel named Full. If time permits, in Exercise 3, you will call another Web service for a hotel named Empty, and then retrieve information regarding room availability.

The Web service for the Full hotel is named Hotel_Full.dll. The Web service for the Empty hotel is named Hotel_Empty.dll. There are five methods in each service.

- **Reserve** takes room types and start and end dates and returns a Boolean value that indicates whether a room is available.

```
<WebMethod()>public Function Reserve (
    strRoomType1 As String, strRoomType2 As String,
    dtmStartDate As Date, dtmEndDate As Date) As Boolean
```

- **Price** returns a double value that is the price of the hotel for one night.

```
<WebMethod()>public Function Price(
    strRoomType1 As String) As Double
```

- **Description** returns a string that describes the hotel.

```
<WebMethod()>public Function Description() As String
```

- **Room** returns a string that describes the rooms of the hotel.

```
<WebMethod()>public Function Room() As String
```

- **Food** returns a string that describes the food available at the hotel.

```
<WebMethod()>public Function Food() As String
```

✍ To call the Full hotel Web service

1. Open the hotel_reservation.aspx file.
2. Add three **label** Web controls to the page with the following IDs: **lblDescription**, **lblRoom**, and **lblFood**. Place them before the drop-down list (just before the line: ` Select a Room Type:
`).
3. If you are using Visual Studio .NET, add a reference to the Full hotel Web service:
 - a. Right-click **ASPNET** in the Solution Explorer, and then click **Add Web Reference**.
 - b. In the **Add Web Reference** dialog box, type `http://localhost/ASPNET/Hotel_Full.asmx` in the **Address** field, press **ENTER**, and then click **Add Reference**.

4. Create a **Page_Load** event procedure.
- a. If you are using Visual Studio .NET, create a proxy variable for the Full hotel named **hotel_FullProxy** of type **ASPNET.localhost.hotel_Full** and then create an instance of the proxy object.

Your code should look like the following:

```
Dim hotel_FullProxy As ASPNET.localhost.hotel_Full
hotel_FullProxy = New ASPNET.localhost.hotel_Full()
```

- b. If you are using Microsoft Notepad, create a proxy variable for the Full hotel named **hotel_FullProxy** of type **myHotel_Full.hotel_Full** and then create an instance of the proxy object.

Note The proxy has already been created and compiled into the `hotel_Full.dll` DLL and placed in the `\bin` directory. The namespace for the Web service is `myHotel_Full`.

Your code should look like the following:

```
Dim hotel_FullProxy As myHotel_Full.hotel_Full
hotel_FullProxy = New myHotel_Full.hotel_Full()
```

- c. Call the **Description**, **Room**, and **Food** methods of the Full hotel's Web service to read information about the hotel. Display the returned information in the corresponding label controls, as shown in the following table.

Label	Method
lblDescription	Description
lblRoom	Room
lblFood	Food

Your code should look like the following:

```
lblDescription.Text = hotel_FullProxy.Description()
lblRoom.Text = hotel_FullProxy.Room()
lblFood.Text = hotel_FullProxy.Food()
```

⚡ **To save and test your work**

1. Save your changes to the file hotel_reservation.aspx.
2. If you are using Visual Studio .NET, build the project by clicking **Build** on the **Build** menu.
3. Using Internet Explorer, go to the hotel reservation page of the ASPNET Web site by viewing http://localhost/ASPNET/hotel_reservation.aspx.
4. You should see information displayed about the hotel, as shown in the following illustration.

Hotel Description:
 This hotel is located in the heart of the city. It is near wonderful shopping and many art museums.

Room Description:
 Each room has a private bath, a mini bar, and a color TV. Double rooms have a balcony and a view of the city.

Restaurant Description:
 There are three restaurants in this hotel: a cozy cafe, a lively bar, and an elegant bistro.

Select a Room type:
 Single No Smoking

Starting date:							Ending date:						
< January 2001 >							< January 2001 >						
Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon	Tue	Wed	Thu	Fri	Sat
31	1	2	3	4	5	6	31	1	2	3	4	5	6
7	8	9	10	11	12	13	7	8	9	10	11	12	13

Exercise 2

Creating a Web Service

In this exercise, you will create a Web service for another hotel, hotel_New.

✦ To create the framework of the Web service

1. If you are using Notepad, create a new .asmx file named hotel_New.asmx, and then add the **@WebService** directive, the **Imports** statements, and the Class definition.

Your code should look like the following:

```
<%@ WebService Language="VB" Class="hotel_New" %>
```

```
Imports System.Web.Services
```

```
Imports System
```

```
Class hotel_New
```

```
End Class
```

2. If you are using Visual Studio .NET, create a new Web service project named hotel_New.

Note Visual Studio .NET creates an .asmx file named Service1.asmx with a class named Service1. Change the name of the Service1.asmx file to **hotel_new.asmx**, and then change the name of the Class to **hotel_New**.

3. Copy the five methods from the hotel_Empty.asmx Web service into the hotel_New Web service.

✦ To customize the WebMethods

- In the hotel_New.asmx page, modify the **Price**, **Description**, **Room**, and **Food** methods to return new values.

✦ To save and test your work

1. Save your changes to hotel_New.asmx.
2. If you are using Visual Studio .NET, build the project by clicking **Build** on the **Build** menu.
3. If you are using Microsoft Notepad, open an MS-DOS® command prompt.
 - a. Navigate to the folder InetPub\wwwroot\ASPNET.
 - b. Run the following commands:

```
wsdl /l:vb /n:myHotel_New
http://localhost/ASPNET/hotel_New.asmx?wsdl
```

```
vbc /out:.\bin\hotel_New.dll /t:library /r:System.dll
/r:System.Xml.dll /r:System.Web.Services.dll
hotel_New.vb
```

- Using Internet Explorer, go to the description page of the Web Service.

The Web service is located at http://localhost/ASPNET/hotel_New.aspx if you are using Microsoft Notepad.

The Web service is located at http://localhost/hotel_New/hotel_New.aspx if you are using Visual Studio .NET.

- Test the **Price**, **Description**, **Room**, and **Food** methods.

⚡ To call `hotel_New` from `hotel_reservation.aspx`

- In the `hotel_reservation.aspx` page, call the `hotel_New` Web service instead of the `hotel_Full` Web service.

Note Refer to Exercise 1 of this lab for detailed instructions on how to call a Web service from the `hotel_reservation.aspx` page.

Exercise 3 (If Time Permits)

Calling Another Web Service

In Exercise 1, you called a Web service named `Hotel_Full`. In this exercise, you will call another Web service named `Hotel_Empty`, and then retrieve information regarding room availability.

⚡ To call another Web service

The user selects a hotel on the `Checkout.aspx` page and that hotel is passed to the `hotel_reservation.aspx` page in the `HotelID` variable. The values are 1 for the Full hotel, and 2 for the Empty hotel.

1. At the beginning of the **Page_Load** event procedure in `hotel_reservation.aspx`, read the value of the `HotelID` variable using the `request.Params` function. Store the value in an integer variable named `intHotelID`.

Your code should look like the following:

```
Dim intHotelID As Integer  
intHotelID = CInt(Request.Params("HotelID"))
```

2. Initialize the text in label `lblReservationConfirmation` with an empty string.

3. Test the value of the variable `intHotelID`.
 - a. If the value is 1, move the code to a proxy for the Full hotel, and call methods from that Web service to this location.
 - b. If the value is 2, create a new proxy for the Empty hotel, and call methods from this Web service.

The **Page_Load** event should look like the following:

```

Dim intHotelID As Integer
intHotelID = CInt(Request.Params("HotelID"))

' Initialize the reservation confirmation message
lblReservationConfirmation.Text = ""

' Which hotel is it?
Select Case intHotelID
  Case 1
    ' Create a proxy for the Full hotel
    Dim hotel_FullProxy As myHotel_Full.hotel_Full
    hotel_FullProxy = New myHotel_Full.hotel_Full()
    ' Get the general, room and food descriptions
    lblDescription.Text = hotel_FullProxy.Description()
    lblRoom.Text = hotel_FullProxy.Room()
    lblFood.Text = hotel_FullProxy.Food()
  Case 2
    ' create a proxy for the Empty hotel
    Dim hotel_EmptyProxy As myHotel_Empty.hotel_Empty
    hotel_EmptyProxy = New myHotel_Empty.hotel_Empty()
    ' Get the general, room and food descriptions
    lblDescription.Text = hotel_EmptyProxy.Description()
    lblRoom.Text = hotel_EmptyProxy.Room()
    lblFood.Text = hotel_EmptyProxy.Food()
End Select

```

✍ To save and test your work

1. Save your changes to the file `hotel_reservation.aspx`.
2. Using Internet Explorer, go to `Checkout.aspx` page in the ASPNET Web site, <http://localhost/ASPNET/Checkout.aspx>
3. Click **Hotel Full**.
You should see the descriptions from the Full hotel.
4. Return to the Checkout page, and then click **Hotel Empty**.
You should see the descriptions from the Empty hotel.

✍ To get the availability of the room

On the `hotel_reservation.aspx` page, when the user clicks **Submit**, call the **Reserve** method of the hotel Web service and, if there is an available room, call the **Price** method to determine the price of the room.

1. Inside the **Submit** button's click event procedure (`cmdSubmit_Click`), after the test for the calendar dates, add code to retrieve the **HotelID** variable from the query string.
2. Test the type of hotel. Again, the values are 1 for the Full hotel, and 2 for the Empty hotel.
3. For each of the two cases, create a proxy for the corresponding Web service and call the **Reserve** method.

The **Reserve** method requires the following parameters:

Parameter	Data Type	Description
<code>strRoomType1</code>	String	Indicates if the room should be single or double.
<code>strRoomType2</code>	String	Indicates if the room should be smoking or non-smoking.
<code>dtmStartDate</code>	Date	The starting date.
<code>dtmEndDate</code>	Date	The ending date.

4. Display a message in the **lblReservationConfirmation** label indicating the availability of a room.

If there is an available room, include in the message the type of room that was selected, the dates requested, and the price.

The code that you have added should look like the following:

```

Dim intHotelID As Integer
Dim blnRoomFree As Boolean = False
Dim dblPrice As Double = 0.00

' Obtain HotelID from QueryString
intHotelID = CInt(Request.Params("HotelID"))

' Ask the hotel service if there is an available room
Select Case intHotelID
    Case 1
        ' Create a proxy for the Full hotel
        Dim hotel_FullProxy As myHotel_Full.hotel_Full
        hotel_FullProxy = new myHotel_Full.hotel_Full()
        ' Get the availability of the selected type of room
        ' for the specified dates
        blnRoomFree = hotel_FullProxy.Reserve _
            (1stRoomType1.SelectedItem.Text, _
            1stRoomType2.SelectedItem.Text, _
            calStartingDate.SelectedDate, _
            calEndingDate.SelectedDate)
        ' Get the price
        dblPrice = hotel_FullProxy.Price _
            (1stRoomType1.SelectedItem.Text)
    Case 2
        ' Create a proxy for the Empty hotel
        Dim hotel_EmptyProxy As myHotel_Empty.hotel_Empty
        hotel_EmptyProxy = new myHotel_Empty.hotel_Empty()
        ' Get the availability of the selected type of room
        ' for the specified dates
        blnRoomFree = hotel_EmptyProxy.Reserve _
            (1stRoomType1.SelectedItem.Text, _
            1stRoomType2.SelectedItem.Text, _
            calStartingDate.SelectedDate, _
            calEndingDate.SelectedDate)
        ' Get the price
        dblPrice = hotel_EmptyProxy.Price _
            (1stRoomType1.SelectedItem.Text)
End Select

' code continued on next page

```

```

' code continued from previous page

' Build the confirmation message
If blnRoomFree Then
    lblReservationConfirmation.Text = _
        "Room available. " & _
        "<br><br> Your selection was:<br>" & _
        lstRoomType1.SelectedItem.Text & ", " & _
        lstRoomType2.SelectedItem.Text & "<br>" & _
        "from " & _
        calStartingDate.SelectedDate.ToShortDateString() _
        & " to " & _
        calEndingDate.SelectedDate.ToShortDateString() _
        & "<br><br>" & "The price will be: " & _
        System.String.Format("{0:c}", dblPrice)
Else
    lblReservationConfirmation.Text = _
        "No room available."
End If

```

✍ To save and test your work

1. Save your changes to the file hotel_reservation.aspx.
2. Using Internet Explorer, go to the Checkout.aspx page in the ASPNET Web site, <http://localhost/ASPNET/Checkout.aspx>
3. Click **Hotel Full**.
4. Select a room type and starting and ending dates, and then click **Reserve**.
You should see a message stating that there is no room available.
5. Return to the Checkout.aspx page and click **Hotel Empty**.
6. Select a room type and starting and ending dates, and then click **Reserve**.
You should see a message stating that a room is available and its price should be displayed.

Review

- What Is a Web Service?
- Calling a Web Service from a Browser
- Calling a Web Service by Using a Proxy
- Creating a Simple Web Service by Using Visual Basic
- Creating and Calling a Web Service by Using Visual Studio .NET

-
1. What is the main purpose of Web services?
 2. How do you find an existing Web service?
 3. What are the different methods of calling a Web service?
 4. What is the WSDL contract for a Web service?

5. How do you create methods for a Web service?

6. How do you call methods of a Web service?