Chapter 9

Active Server Pages (ASP.NET)

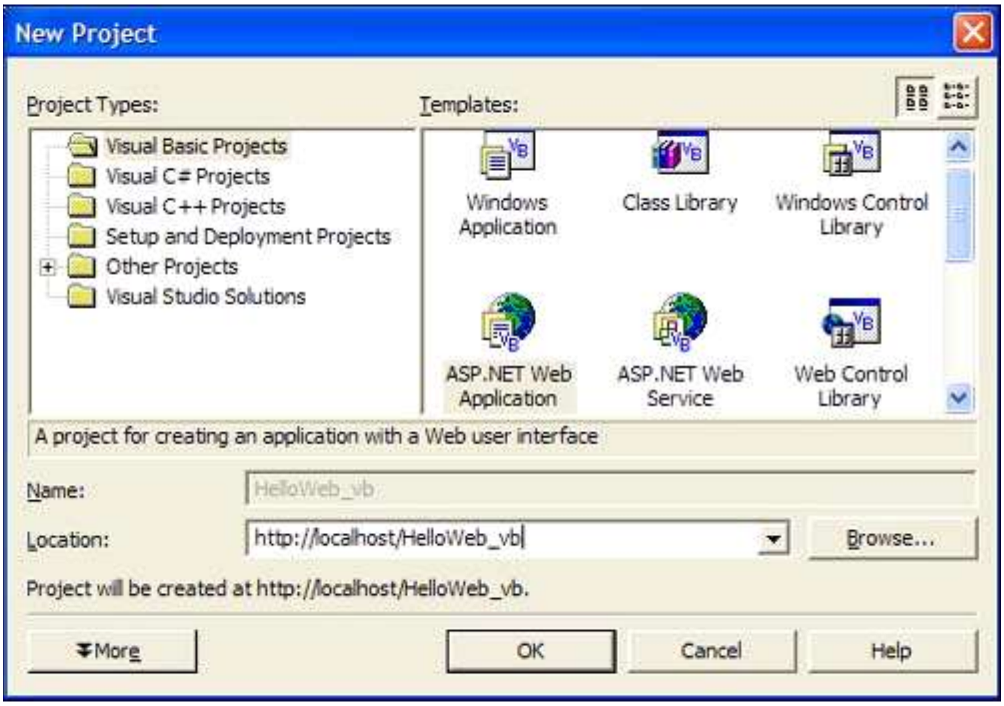
Chapter 9
Active Server Pages (ASP.NET)

ASP.NET is an object-oriented, event-driven platform for writing Web-based applications. Some of the major improvements and benefits that ASP.NET gives you:

- Using the various **caching** methods in ASP.NET
- Using the **Web.Config** configuration file, you can implement robust page- and directory-level security without writing code for each page.
- You have the same **Code Editor** features that Windows Forms applications have.
- **Debugging** ASP.NET applications is just like debugging Windows applications.
- **A lot of controls,** including validation controls, grid controls, and list controls.
- ASP.NET applications are **100% compiled**.

Hello ASP.NET!

In the **New Project** dialog. Select the **ASP.NET Web Application**. Change the name of the application to **http://localhost/HelloWeb_vb** or **http://localhost/HelloWeb_cs**



A project for creating an application with a Web user interface

Name: HelloWeb_vb

Location: http://localhost/HelloWeb_vb

Project will be created at http://localhost/HelloWeb_vb.

Visual Studio creates the **virtual Web directory** at the location you specify in the name of the project. In this case, **localhost**, which is the default name for the local Web server running on your machine. Physically it is created in **\inetpub\wwwroot\helloweb_cs** or **\inetpub\wwwroot\helloweb_vb** directory. In **Visual Studio Projects** folder under **My Documents**, you'll see HelloWeb_vb or HelloWeb_cs directory, containing only the solution file.

The **Webform1.aspx** file has a **Webform1.aspx.vb** and **Webform1.aspx.resx** underneath it in the tree view. The code-behind file has the same name of the ASPX file, but with the .cs or .vb extension. The .resx file is the resource file, which contains localization information for the Form.

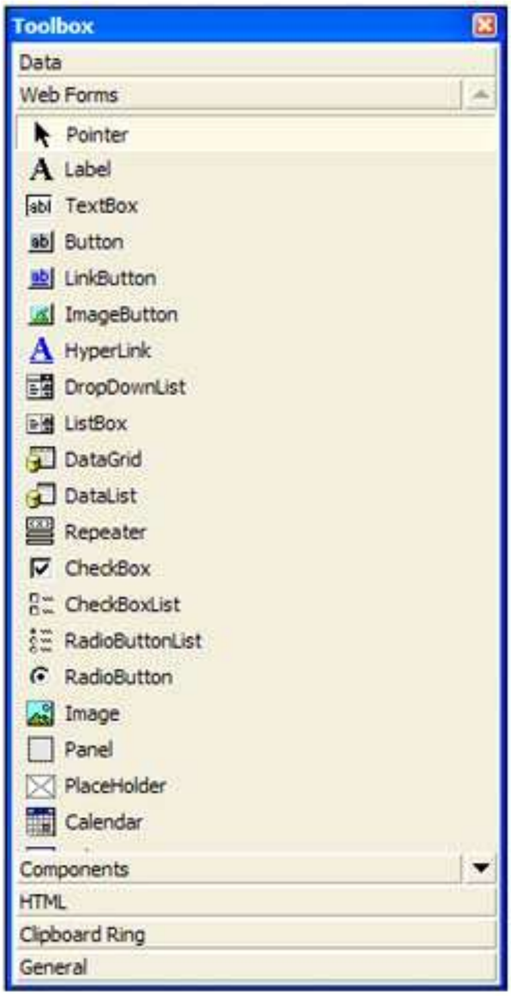
If you click the **HTML button** in the lower left of the Web Forms Designer, and click the **Design button** in the lower-left corner of the HTML designer, you're taken back to the design surface for the ASPX page. To get to the code-behind file, you can double-click the ASPX page, you can press the F7 key, or you can double-click class file in the Solution Explorer.

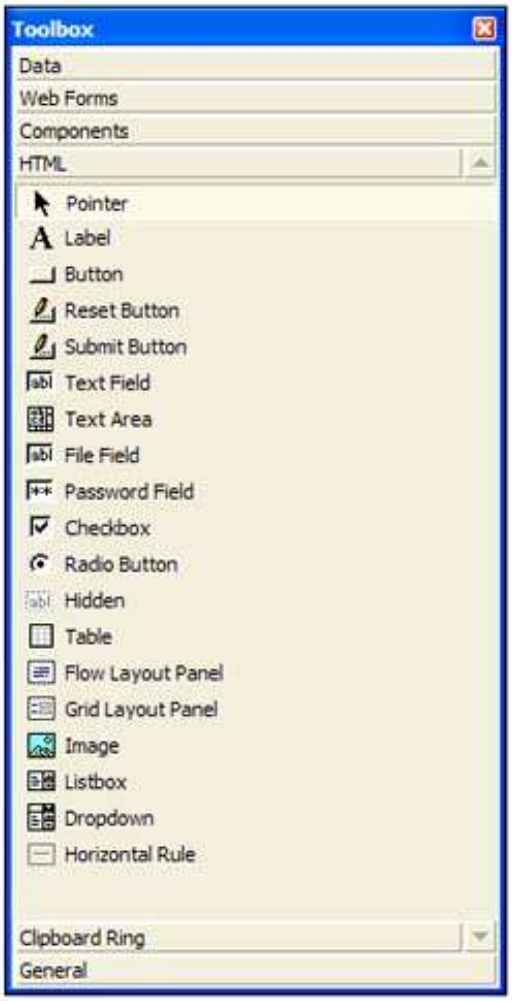
The class file inherits the **System.Web.UI.Page** class. When an ASP.NET application is compiled, the code-behind files for each ASPX page are compiled into a single assembly with a **DLL** extension and are placed in the Bin directory.

Files That Make Up an ASP.NET Web Application

Webform1.aspx	The visual part of the Web Form. You can modify the HTML to pass the user interface of a Web form or drag controls from the Toolbox onto it.
WebForm1.aspx.cs	Code-behind class file that the ASPX file inherits its functionality from.
Web.Config	XML-based configuration file for the project. You can set properties on security, caching, state, and tracing information in **Web.Config** file.
Global.asax	Contains application-level events for an ASP.NET project.
Styles.css	Default style sheet for a project. You can double-click .css files to edit them in the Style Sheet Designer.
AssemblyInfo	Contains assembly-specific information for the project's assembly output.
.vdisco	Discovery file for XML Web services in the application. More about it in : "XML Web Service in .NET".

Adding Controls to a Web Form





Adding controls to a Web Form is like adding to a Windows Form. You visually design a page by dragging predefined controls from Toolbox onto the Web Form.

HTML controls are the same HTML-tag-based controls you use in a regular HTML page. Server controls offer a richer user interface, and have an object model that you can access in the code-behind class files for an ASPX page.

Drag an HTML text field to the ASPX page and then drag a TextBox server control.

If you switch to the HTML view in the designer, you'll see :

```
<body>
   <form id="Form1" method="post" runat="server">
      <P><asp:TextBox id="TextBox1"
runat="server"></asp:TextBox></P>
      <P>&nbsp;</P>
      <P> <INPUT type="text"></P>
   </form>
</body>
```

By adding the **runat=server** attribute to any control, be it an HTML control or a server control, you can access the control from the inherited **code-behind file**.

C# **protected System.Web.UI.<u>WebControls.TextBox</u> TextBox1;**

Notice that the HTML Input control isn't declared in the code-behind file.
Switch to the HTML view of the WebForm1.aspx page and modify the HTML Input control to look like this:

```
    <INPUT type="text" runat="server" id="HtmlTextBox">
```

Then **save the page** and switch back to the code-behind class.
You'll notice this declaration:

C# **protected System.Web.UI.<u>HtmlControls.HtmlInputText</u> HtmlTextBox;**

what's the difference, and why would you use HTML controls instead of server controls?

- Server controls don't expose **client events**; HTML controls do. In other words, you can't use client-side JavaScript or VBScript with a control that has a runat=server attribute.
- Server controls offer a **rich object model** in the code-behind class that enables you to program Web applications in a way that's similar to writing Windows applications—by responding to events and setting properties.
- Both sets of controls offer **auto-complete** and **auto-list members** in the HTML view, so that functionality isn't specific to one type of control.
- There's overhead associated with using server controls. Because server controls must be part of the compiled assembly and processed by the ASP.NET runtime on the server, it isn't always as **efficient** as using a straight HTML control.

Any HTML element can have a runat=server attribute !!

Example, let's say you have an HTML table, and you need to change the color of a table cell, based on something that occurs in a server event. If you add the runat=server attribute to the TD HTML element and give it an ID, the code-behind class will have the following declaration:

```
Protected WithEvents TD1 As
System.Web.UI.HtmlControls.HtmlTableCell
```

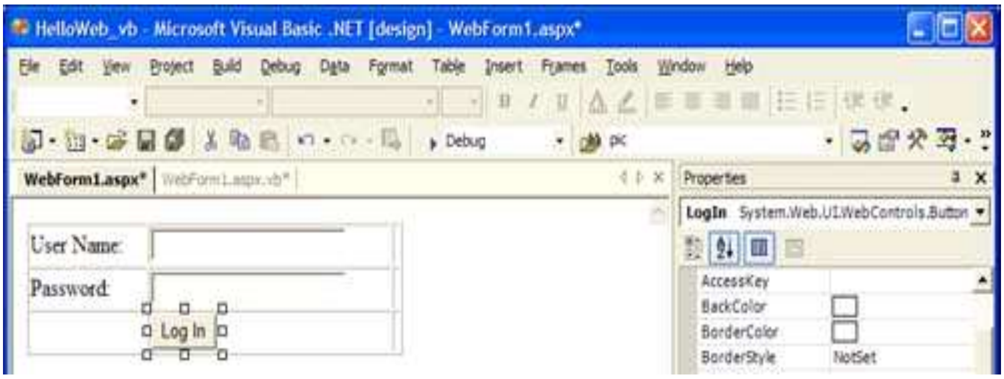
In your code, you can then use a Select Case to set Style attributes on the HtmlTableCell object:

```
Select Case Request("AlertStatus")
    Case "Red"  :
    TD1.Attributes.Add("background", "red.jpg")
    Case "Yellow"  :
    TD1.Attributes.Add("background", "yellow.jpg")
    Case Else  : TD1.Attributes.Add("background",
    "green.jpg")
End Select
```

Responding to Server Control Events

Steps to modify the WebForm1.aspx page, to write a simple **login form**..

1. Make sure that you're in the Web Forms Designer.
2. Delete the HTML text field control and the TextBox server control from the form.
3. From the **Table** menu, select **Insert**, **Table**.
4. Click the OK button to add default table with 3 columns and 3 rows.
5. Drag two **Label** controls from the **Web Forms** to the first two rows in the first column of the table. Change their **Text** property to User Name and Password, respectively.
6. Drag two **TextBox** controls to the first two rows in the second column of the table. Change their **Name** property to UserName and Password, respectively.
7. Drag a **Button** control to the third row on the second column in the table, and change its **Text** property to Log In and change its **Name** property to LogIn.


HelloWeb_vb - Microsoft Visual Basic .NET [design] - WebForm1.aspx*
File Edit View Project Build Debug Data Format Table Insert Frames Tools Window Help
Debug
WebForm1.aspx*
WebForm1.aspx.vb*
User Name
Password
Log In
Properties
LogIn System.Web.UI.WebControls.Button
AccessKey
BackColor
BorderColor
BorderStyle
NotSet

By default, the Web Forms Designer defaults to **absolute positioning** and **grid layout** .
Absolute positioning sets the **x** and **y** coordinates of each control using the **Style** property.
I'm accustomed to using **Table** to position my HTML elements.
To use **FlowLayout**, right-click ASPX page and set its **Page Layout** property to **FlowLayout**.

Double-click on the Log In button to get to its Click event, and add the following code, which writes out the contents of the TextBoxes to the form.

```
private void LogIn_Click(object sender, System.EventArgs e)
{
        Response.Write(UserName.Text);
        Response.Write("<br>");
        Response.Write(Password.Text);
        Response.End();
}
```

Right-click the WebForm1.aspx page in the Solution Explorer and select **Build and Browse**, the page will be compiled and displayed in the internal browser of Visual Studio .NET.

ASP.NET pages run from the compiled assembly in the Bin directory. So, if you make any changes to your code-behind files, you must **rebuild** the solution to make sure that you're working with the current version of the files.
Changes to standard HTML are not compiled until the page is accessed, so you can select **Save All Files** from File menu to save HTML changes have been (no need to rebuild the project)

After the page is displayed in the browser, enter values in the UserName text box and the Password text box and click the Log In button.

Using Validation Controls

When you validate a control, you're checking whether a control has data in it and conforms to a specific pattern, (such as an email address), or you're checking the range of data entered.

Five validation controls in Toolbox that you can drag to a form and associate with a control.

RequiredFieldValidator	Forces the user to enter a value into the specified control.
CompareValidator	Compares a user's entry against a constant value, or against a property value of another control, using a comparison operator.
RangeValidator	Checks user's entry is between specified lower and upper boundaries. Ranges are pairs of **numbers**, **alphabetic characters**, **dates**.
RegularExpression Validator	Checks that the entry matches a pattern defined by a regular expression.

Each validatior has **Text** (displays text) and **ErrorMessage** property (displays if an error occurs).
RequiredFieldValidator has a **ControlToValidate** which takes ID of a valid control on the form.
Add, **Add Web Form**. change the name from WebForm2.aspx to **Success.aspx**,

Double-click `Success.aspx` to get to the **`Form_Load`** event for the page, and add the code :

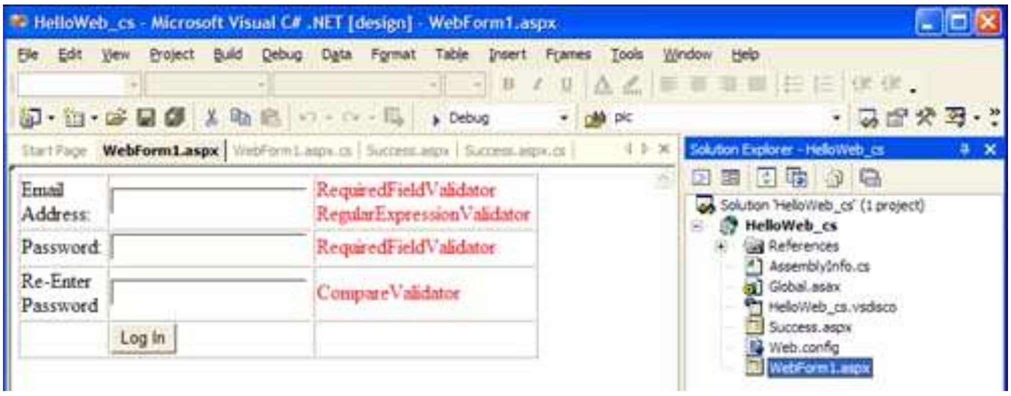
```
private void Page_Load(object sender, System.EventArgs e)
        {
           Response.Write("<H1>You are now logged in</H1>");
        }
```

To modify the WebForm1.aspx page, you must do the following:

1. Add a new table row above the Log In button.
   by clicking inside the cell that contains the Log In button, and then right-clicking and selecting Insert, Row Above from the contextual menu.
2. Drag a Label and TextBox. Change the **Text** of the Label to

Re-Enter Password, and change the **ID** property of the TextBox to Password2.

3. Change the **Text** of the Label control that says UserName: to now say Email Address:.
4. Drag a RequiredFieldValidator, and place it in the column next to the Username TextBox.
5. Drag a RegularExpressionValidator and place it next to the RequiredFieldValidator.
6. Drag a RequiredFieldValidator and place it in the column next to the Password TextBox.
7. Drag a CompareValidator and place it in the column next to the Password2 TextBox.


HelloWeb_cs - Microsoft Visual C# .NET [design] - WebForm1.aspx
File Edit View Project Build Debug Data Format Table Insert Frames Tools Window Help
Debug
pic
Start Page WebForm1.aspx WebForm1.aspx.cs Success.aspx Success.aspx.cs
Email Address:
RequiredFieldValidator
RegularExpressionValidator
Password:
RequiredFieldValidator
Re-Enter Password
CompareValidator
Log In
Solution Explorer - HelloWeb_cs
Solution 'HelloWeb_cs' (1 project)
HelloWeb_cs
References
AssemblyInfo.cs
Global.asax
HelloWeb_cs.vsdisco
Success.aspx
Web.config
WebForm1.aspx

The next step is to set the properties on the validation controls.

1. Select the RequiredFieldValidator1 control, and change the **ControlToValidate** property to UserName. Then change the **ErrorMessage** property to "Email Address Required".
2. Select RegularExpressionValidator. In **ValidationExpression**, click ellipses (…) button to get Regular Expression Editor. Select Internet Email Address and click OK. Change **ErrorMessage** to "Invalid Email Format". Change **ControlToValidate** to Username.
3. Select the RequiredFieldValidator2 control, and change the **ControlToValidate** property to Password. Then change the **ErrorMessage** property to "Password Required".
4. Select CompareValidator1, and change **ControlToCompare** to Password. Change **ControlToValidate** to Password2. Change **ErrorMessage** to "Passwords do not match".

Rename the WebForm1.aspx to **Login.aspx**. By right-clicking it and selecting Rename.

Now, double-click the Log In button, and add the code:

```
private void LogIn_Click(object sender,
System.EventArgs e)
{
  if (UserName.Text == "jason@mams.net" &&
Password.Text == "password")
      Response.Redirect(@"Success.aspx");
}
```

To make user interface more friendly, you can set **InitialValue** of RequiredFieldValidator to a red asterisk . You should also set **TextMode** of Password and Password2 text boxes to Password so that the data entered into the field is filled with asterisks.

Build. Enter text that isn't email address, and different passwords in Passwords textboxes.

Validation occurs on the client and the page isn't posted to the server until the data is correct.

To display a summary of validation errors, you can use the ValidationSummary control on a page. This takes all the validation errors on the page, and places them in a nice bulleted list. You can use the **ShowMessageBox** property set to True.

To validate a page in **server-side code**, you check the **IsValid** property of a page.

- by setting the **EnableClientScript** property to False for each validation control.
- In the Page_Load event checks the IsValid property to check controls on the page .

```
If Page.IsValid Then
   If UserName.Text = "jason@mans.net"  And
  Password.Text = "password" Then
          Response.Redirect("Success.aspx")
   Else
          LogIn.Text = "Invalid Data, please retry"
   End If
End If
```

Managing State in ASP.NET Web Applications

- The Internet is a stateless application. The challenge is to keep track of where the user is in your Web application, and where to take his next. This is done by maintaining state.
- you keep state on the **client** or on the **server**.

Managing State on the Client (page-level state in the browser).

1. Using Cookies

- Cookies are a key-value paring of collection data that are saved on the client's hard drive.
- If client closes the browser and revisits your site, you check for the cookie in a Page_Load.
- You can modify **Web.Config** to use **sessionless** cookies, i.e. cookie data is encrypted and passed in the query string of the browser, so no data is actually stored on user's machine.

```
<sessionState
   mode="InProc"
   stateConnectionString="tcpip=127.0.0.1:42424"
   sqlConnectionString="data
source=127.0.0.1;user id=sa;password="
   cookieless="true"
   timeout="20"
/>
```

- To access cookies in code, you access **Cookies** of the **Request** or **Response** object

```
Dim cookie As HttpCookie =
Request.Cookies("UserID")
If cookie Is Nothing Then
        Response.Cookies("UserID").Value =
      "Jason"
        Response.Cookies("UserID ").Expires =
      "January 1, 2010"
        Response.Cookies("UserID ").Path = "/"
Else
        Response.Write(Request.Cookies("UserID")
End If
```

You must set **Expires** to save the cookie, else the cookie is deleted when session ends.

2. Understanding View State

- Each control and ASPX page has an **EnableViewState** property (True by default).
- When a page is processed that has ViewState enabled, a hidden field named **Viewstate** is added to the ASPX page and all the page data for the form is preserved in it.
- ViewState is a page-level state management option.
- Because the page data is kept in the hidden control, the page size can grow.
- You can set ViewState data in code, similar to a Cookies or Session object, as:

```
Viewstate("UserID") = "Bob"
```

3. Using the HTML Hidden Control

- Using hidden fields is a common way to store information in ASP.
- You can set the Value property of the control to store page-specific information.

4. Using the Query String

- HTTP-GET is set using the Method=Get attribute on **Form** tag in the HTML of your page, you automatically send all the data in the fields through the query string of the browser.
- When you pass values of a form using HTTP-GET, the initial control ID is separated from the requested page by an ampersand, and the remaining fields are separated by the question mark character. If you were to pass the data to the Success.aspx page using HTTP-GET in the code you wrote for the Login.aspx page, the query string is: **http://localhost/helloweb_vb?Username=jason@mans&Password=password**

Managing State on the Server

To manage state on the server side, you use either session state or application state.

1. Using Session State

Using session-level state enables you to track variable data for a user throughout his visit.
Using the **HttpSessionState** class, same key-value syntax that you use for ViewState.
Each time a browser hits your site, IIS creates a unique session ID for the browser session if you access the Session object in code. If you don't reference the Session object in code, IIS doesn't create a unique ID for the end user's session on your site. Session information is useful to save data between multiple page calls. For example, when I visit my bank's Web site, I must enter my username and password every time I visit. But after I log in to the site, each page I navigate knows who I am, so the data is being passed to all pages for my session through the use of session state. To set or retrieve values for session state, you use this syntax:

```
If Session("UserID") = "Bob" then ...

Reponse.Write(Session("UserID"))
```

After the browser is closed or the end user navigates to another site, session data is lost. When user returns to site, he must create new session data based on current session.

Web.Config file gives you options as to where session-level state can be stored for a site. By default, session state is stored in the same process as IIS. You can store session state in SQL Server. By modifying **mode** attribute in the sessionState section :

```
<sessionState
mode="SqlServer"
stateConnectionString="tcpip=127.0.0.1:42424"
sqlConnectionString="data
source=127.0.0.1;user id=sa;password="
cookieless="false"
timeout="20"
/>
```

You must also supply the correct authentication information, and run a special SQL script that creates database and temporary tables in SQL Server to hold the state data.

In **Global.asax** file, you can write code in **Session_OnStart** and **Session_OnEnd** events. This ensures that a session-specific event, such as updating a hit counter in a database, is occurring each time a session begins or ends.

2. Using Application State

- Application-level state is the top level in the state management hierarchy. The first time someone accesses your Web site, the **Application** object for the site is created. The Application object is alive until the server is rebooted, IIS is restarted, or a new copy of the Web site is deployed. In the

Global.asax file, there are **Application_OnStart** and **Application_OnEnd** events that you can write code to respond to; they're similar to the Session_OnStart and Session_OnEnd events. The difference is that application-level variables are global for all sessions for your Web site, not for individual users.

What Are XML Web Services?

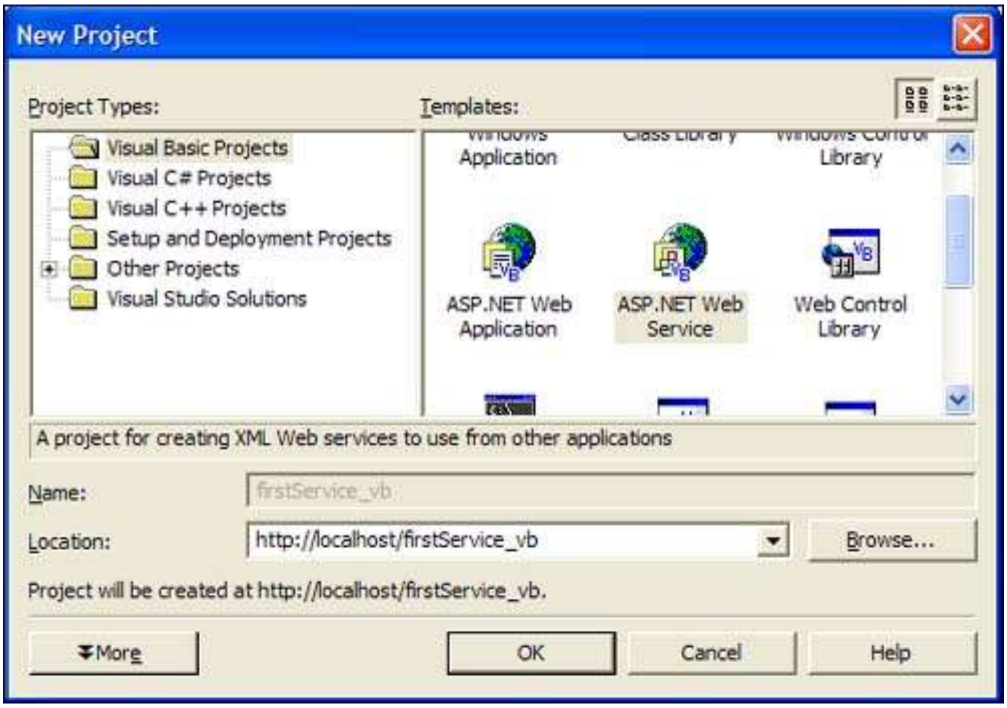
XML Web services are application components that can expose functionality over hypertext transfer protocol (HTTP) using the simple object access protocol (SOAP). Using Web services, you have the ability to expose methods in your applications to anyone on the Internet.

Web services description language (**WSDL**) is an XML-based contract that defines what methods are contained in a Web service.

DISCO, or discovery, files provide a mechanism to discover what Web services are available at a particular site. If you have a Web site and you want to expose a number of Web services, you can create a DISCO file that can be queried by others to find out what your site offers. Discovery files aren't required to make a Web service work; they just provide a mechanism to find out what's available on a specific Web site.

Universal description, discovery, and integration (UDDI) is a distributed repository for Web services. The global community of the Internet needed a mechanism for people to find what Web services are available. You can go to http://www.uddi.org and search for Web services by type, functionality, industry, or company.

Creating Your First Web Service



A project for creating XML Web services to use from other applications

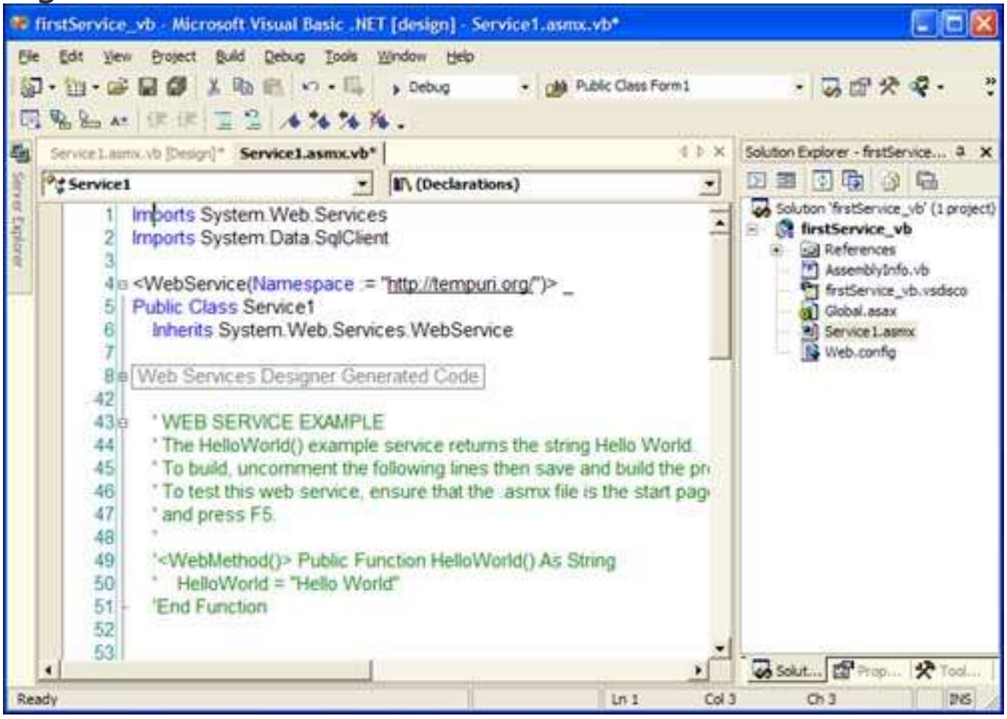
Name: firstService_vb

Location: http://localhost/firstService_vb

Project will be created at http://localhost/firstService_vb.

From this point, you can modify properties, add items from the toolbox, or switch to the Code view. If you look at the Properties window for this new Web service, you'll notice the name isn't firstService, it's Service1. Service1 is the default name for all new Web services added to a project. When you created the project firstService, you created the Internet Information Server (IIS) virtual directory named firstService to contain the Web service named Service1. For now, you can leave the name Service1 as it is. On the designer, click the link that switches you to Code view.

Figure :The Code view of an XML Web service.


firstService_vb - Microsoft Visual Basic .NET [design] - Service1.asmx.vb*
File Edit View Project Build Debug Tools Window Help
Debug
Public Class Form1
Service1.asmx.vb [Design]*
Service1.asmx.vb*
Service1
(Declarations)
Solution Explorer - firstService...
Solution 'firstService_vb' (1 project)
firstService_vb
References
AssemblyInfo.vb
firstService_vb.vsdisco
Global.asax
Service1.asmx
Web.config
Solut...
Prop...
Tool...
Ready
Ln 1
Col 3
Ch 3
INS

```
Imports System.Web.Services
Imports System.Data.SqlClient

<WebService(Namespace := "http://tempuri.org/")> _
Public Class Service1
    Inherits System.Web.Services.WebService

Web Services Designer Generated Code

    ' WEB SERVICE EXAMPLE
    ' The HelloWorld() example service returns the string Hello World
    ' To build, uncomment the following lines then save and build the pr
    ' To test this web service, ensure that the .asmx file is the start pag
    ' and press F5.
    '
    '<WebMethod()> Public Function HelloWorld() As String
    '    HelloWorld = "Hello World"
    ' End Function
```

You're now in the code-behind file for the Web service named Service1. Web services in Visual Studio .NET have a .ASMX extension, so naturally the code-behind would be either ASMX.VB (for Visual Basic .NET) or ASMX.CS (for C#). It follows the same pattern as ASPX pages in Web applications. The code-behind file simply tacks on the language extension to the designer file.
If you look at the code in the class file, you'll notice that the System.Web.Services namespace is imported into this class.

Classes of the System.Web.Services Namespace

Class	Description
WebMethodAttribute	Adding this attribute to a method within an XML Web service created using ASP.NET makes the method callable from remote Web clients
WebService	The optional base class for XML Web services, which provides direct access to common ASP.NET objects, such as application and session state
WebServiceAttribute	Used to add additional information to an XML Web service.
WebService BindingAttribute	Declares the binding one or more XML Web service methods implemented within the class implementing the XML Web service

Simply importing the System.Web.Services namespace doesn't make this a Web service. The class file must inherit the WebService class. So, in the Service1.asmx file, the following code makes this class a Web service:

VB.NET

```
Imports System.Web.Services
<WebService(Namespace := "http://tempuri.org/")> Public Class
Service1
    Inherits System.Web.Services.WebService
```

C#

```
using System.Web.Services;
namespace firstService_cs
{
   public class Service1 : System.Web.Services.WebService
```

All this is done automatically by Visual Studio .NET when you create a new Web service project or you add an ASMX file to your application.

The Namespace attribute points to http://www.tempuri.org/. This is the default namespace for all new Web services created with Visual Studio .NET. You should change it before you deploy your Web service application. The namespace defines what data is in the Web service and what rules the data should follow. It's the same concept that you learned about yesterday when you created XML schema definition (XSD) files for the XML files in your project. In that case, the namespace defined what the XML should look like; the namespace for a Web service defines what the Web service should look like.

Notice the commented-out HelloWorld method in the Service1 class. This sample method is in every new ASMX file in Visual Studio .NET. The HelloWorld method looks like any other method in the class files you worked with until today, with the exception of the WebMethod attribute. By prefixing a method name with the WebMethod attribute, it becomes available to the outside world.

Note
Attributes are tags that enable you to further define objects in an application. The attribute information is then examined at runtime through reflection, letting the language-specific compiler determine what to do with the attribute information. In the case of the WebMethod attribute, the ASP.NET engine knows to make the method available to remote callers. In C#, attributes are enclosed in brackets, *[attributename]*, and in Visual Basic .NET, they're enclosed by less-than/greater-than signs, <*attributename*>.

To make the HelloWorld method available to the outside world, uncomment the method. Make sure that you also uncomment the WebMethod attribute. Before you run the application, you must add two more methods that you'll consume later—it'll also give you an idea of how to use Web services in a real-world scenario.

Listing has two methods: GetCustomers and GetCustomerByID. Both use ADO.NET that you learned about over the past few days. Add these methods to the class file for Service1.asmx.

VB.NET

```
<WebMethod()> Public Function GetCustomers() As DataSet

   Dim cn As New SqlConnection(
"Server=jb1gs\NetSDK;Database=pubs;" _
           & "Integrated Security=SSPI")

   Dim da As SqlDataAdapter = New SqlDataAdapter ("SELECT *
from Authors", cn)

   Dim ds As DataSet = New DataSet()
   da.Fill(ds, "Customers")

   Return ds

End Function

<WebMethod()> Public Function GetCustomerByID (ByVal ID As
String) As DataSet

   Dim cn As New SqlConnection(
"Server=jb1gs\NetSDK;Database=pubs;" _
           & "Integrated Security=SSPI")

   Dim da As SqlDataAdapter = New SqlDataAdapter _
         ("SELECT * from Authors where au_id = ?", cn)

   da.SelectCommand.Parameters.Add("au_id", ID)
   Dim ds As DataSet = New DataSet()
   da.Fill(ds, "Customers")
   Return ds
End Function
```

C#

```
[WebMethod]
public DataSet GetCustomers()
{
      SqlConnection cn = new SqlConnection
        (@"Server=jb1gs\NetSDK;Database=Northwind;Integrated
Security=SSPI");
      SqlDataAdapter da  = new SqlDataAdapter("SELECT * from
Customers", cn);
      DataSet ds = new DataSet();
      da.Fill(ds, "Customers");
      return ds;
}

[WebMethod]
public DataSet GetCustomerByID(string ID)
{
      SqlConnection cn = new SqlConnection
        (@"Server=jb1gs\NetSDK;Database=Northwind;Integrated
Security=SSPI");

      SqlDataAdapter da  = new SqlDataAdapter
        (@"SELECT * from Customers where CustomerID =
@CustomerID", cn);

      da.SelectCommand.Parameters.Add("@CustomerID",
      SqlDbType.VarChar, 15).Value = ID;

      DataSet ds = new DataSet();
      da.Fill(ds, "Customers");
      return ds;
}
```

Note

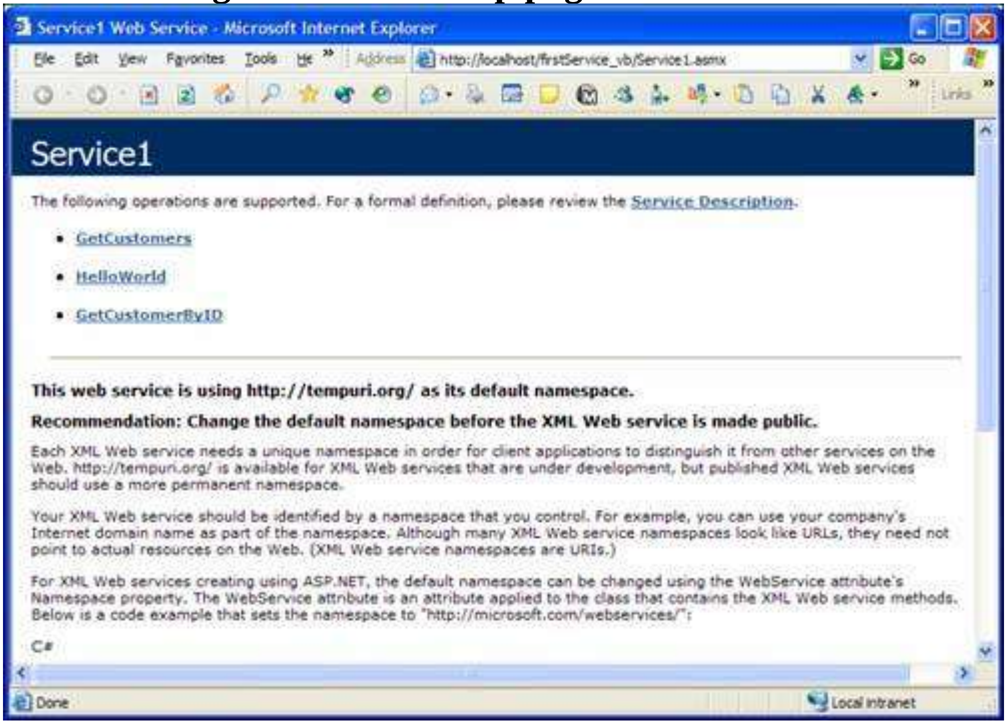
The HTTP-POST and SOAP protocols are enabled by default, but not the HTTP-GET protocol. This was done for security reasons. To enable HTTP-GET for your Web Services, and to successfully do the exercises today, you need to modify the Web.Config file to allow the HTTP-GET protocol to be used. To do this, open the Web.Config file, and add the following section to after the <globalization> tag at the end of the Web.Config file:

```
<webServices>
   <protocols>
      <add name="HttpGet"/>
   </protocols>
</webServices>
```

You can also enable and disable specific protocols in the machine.config file, which will affect all projects for an entire machine. To learn more about the configuration files, and how they can affect your applications, look up Configuration Options under XML Web Services in the Dynamic Help file.

Press F5 to run the application. The browser now opens with the auto-generated Service Help Page, as shown in next figure .

Figure . Service Help page for a Web service.


Service1 Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools He
Address http://localhost/firstService_vb/Service1.asmx
Go
Links

Service1

The following operations are supported. For a formal definition, please review the Service Description.

- GetCustomers
- HelloWorld
- GetCustomerByID

This web service is using http://tempuri.org/ as its default namespace.

Recommendation: Change the default namespace before the XML Web service is made public.

Each XML Web service needs a unique namespace in order for client applications to distinguish it from other services on the Web. http://tempuri.org/ is available for XML Web services that are under development, but published XML Web services should use a more permanent namespace.

Your XML Web service should be identified by a namespace that you control. For example, you can use your company's Internet domain name as part of the namespace. Although many XML Web service namespaces look like URLs, they need not point to actual resources on the Web. (XML Web service namespaces are URIs.)

For XML Web services creating using ASP.NET, the default namespace can be changed using the WebService attribute's Namespace property. The WebService attribute is an attribute applied to the class that contains the XML Web service methods. Below is a code example that sets the namespace to "http://microsoft.com/webservices/":

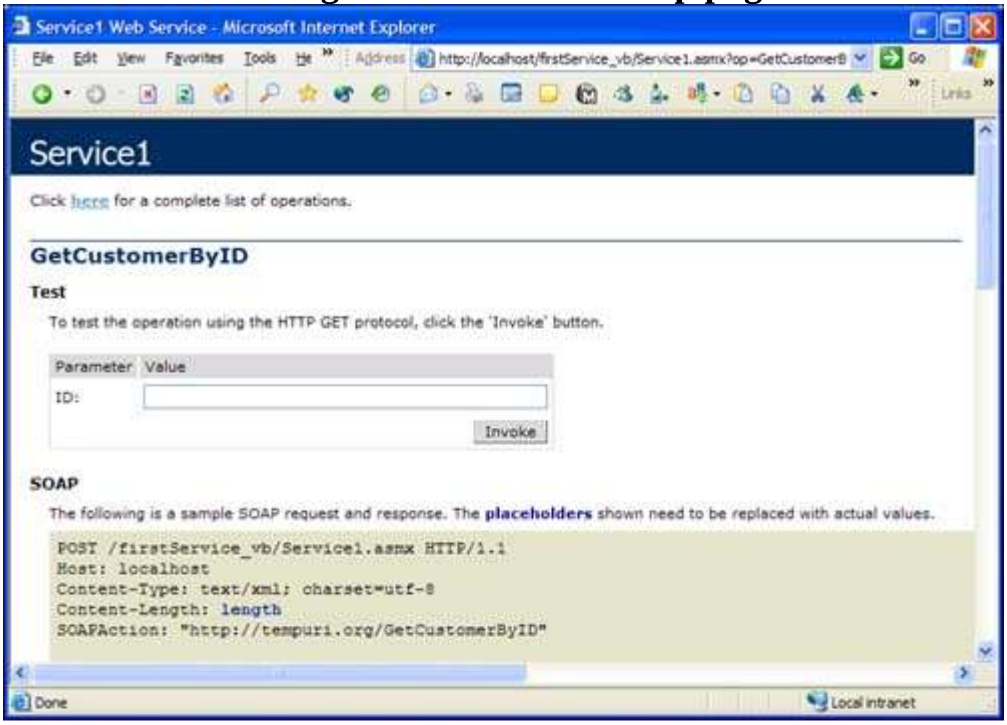
C#

The Service Help Page provides you with a list of all the methods that are available in the Web service and some sample code that shows the implementation of the Web service in C# and Visual Basic .NET. Any time you reference an ASMX file without any parameters, you get the Service Help Page. Notice that the three methods you added to the class file are listed. If you didn't include the WebMethod attribute in one of the method declarations, you won't see it in the list.

The URL that the browser is pointing to is simply the name of the server, the project name, and then the ASMX filename.

If you click the GetCustomerByID link, you're taken to the Service Description page. The Service Description page gives you the WSDL grammar of how to access this method via SOAP, HTTP-GET, or HTTP-POST. That's a nice feature, but you really don't need to know any of that if you're using Visual Studio .NET to consume the Web service. Figure is what the Service Help page looks like for the GetCustomerByID method.

Figure . The Service Help page.


Service1 Web Service - Microsoft Internet Explorer
File Edit View Favorites Tools He
Address http://localhost/firstService_vb/Service1.asmx?op=GetCustomerB
Go
Links
Done
Local intranet

Service1

Click here for a complete list of operations.

GetCustomerByID

Test

To test the operation using the HTTP GET protocol, click the 'Invoke' button.

Parameter	Value
ID:	

Invoke

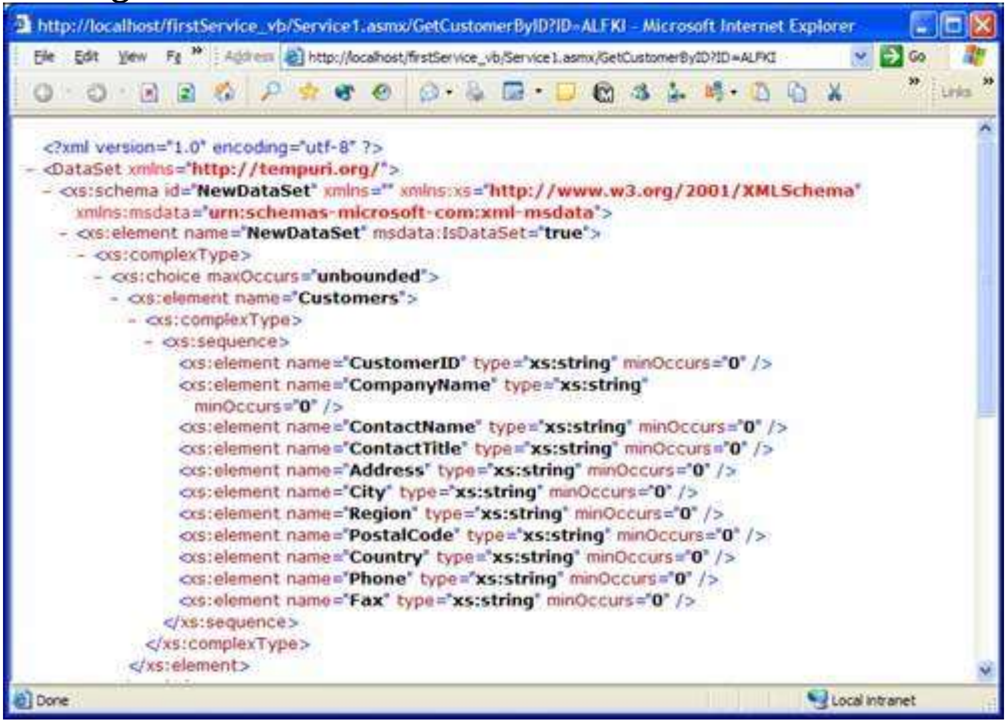
SOAP

The following is a sample SOAP request and response. The **placeholders** shown need to be replaced with actual values.

```
POST /firstService_vb/Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/GetCustomerByID"
```

Notice that because the method accepts a parameter, you're given a text box to fill in the CustomerID parameter. If your Web service methods accept parameters, there will be a text box for each parameter. This is a great feature that makes it very easy to test your methods before you deploy your Web services.
For the CustomerID method, type in **ALFKI**. Now, you can click the Invoke button to actually run the GetCustomerID method.
The results are an XML file that's returned through the browser, as next Figure demonstrates.

Figure XML returned from the HelloWorld Web service.



http://localhost/firstService_vb/Service1.asmx/GetCustomerByID?ID=ALFKI - Microsoft Internet Explorer
File Edit View Fa
Address http://localhost/firstService_vb/Service1.asmx/GetCustomerByID?ID=ALFKI
Go
Links
Done
Local intranet

```
  <?xml version="1.0" encoding="utf-8" ?>
- <DataSet xmlns="http://tempuri.org/">
  - <xs:schema id="NewDataSet" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    - <xs:element name="NewDataSet" msdata:IsDataSet="true">
      - <xs:complexType>
        - <xs:choice maxOccurs="unbounded">
          - <xs:element name="Customers">
            - <xs:complexType>
              - <xs:sequence>
                  <xs:element name="CustomerID" type="xs:string" minOccurs="0" />
                  <xs:element name="CompanyName" type="xs:string"
                    minOccurs="0" />
                  <xs:element name="ContactName" type="xs:string" minOccurs="0" />
                  <xs:element name="ContactTitle" type="xs:string" minOccurs="0" />
                  <xs:element name="Address" type="xs:string" minOccurs="0" />
                  <xs:element name="City" type="xs:string" minOccurs="0" />
                  <xs:element name="Region" type="xs:string" minOccurs="0" />
                  <xs:element name="PostalCode" type="xs:string" minOccurs="0" />
                  <xs:element name="Country" type="xs:string" minOccurs="0" />
                  <xs:element name="Phone" type="xs:string" minOccurs="0" />
                  <xs:element name="Fax" type="xs:string" minOccurs="0" />
                </xs:sequence>
              </xs:complexType>
            </xs:element>
```

Notice that the URL has the name of the ASMX file and then the method name you're trying to call:
http://localhost/firstService_vb/Service1.asmx/HelloWorld

This is the standard syntax to reference a method in a Web service. If the method expects parameters, you use the same query string syntax that you use for any URL that expects parameters. As an example, the following code assumes that a parameter named ID and a parameter named ZipCode are being passed to the HelloWorld method:

http://localhost/firstService_vb/Service1.asmx/HelloWorld?ID=1&
ZipCode=33486
The next step is to consume the Web service from a client application and do something useful with the XML that's returned.

Consuming XML Web Services

Every time we come upon XML in this book, I mention that you don't have to know about XML to get anything done with XML. That still holds true. Although an XML Web service returns XML, the controls, classes, and tools in Visual Studio .NET natively read and write XML, so you don't need to understand the XML part—you only need to understand how to get the data from the XML part.

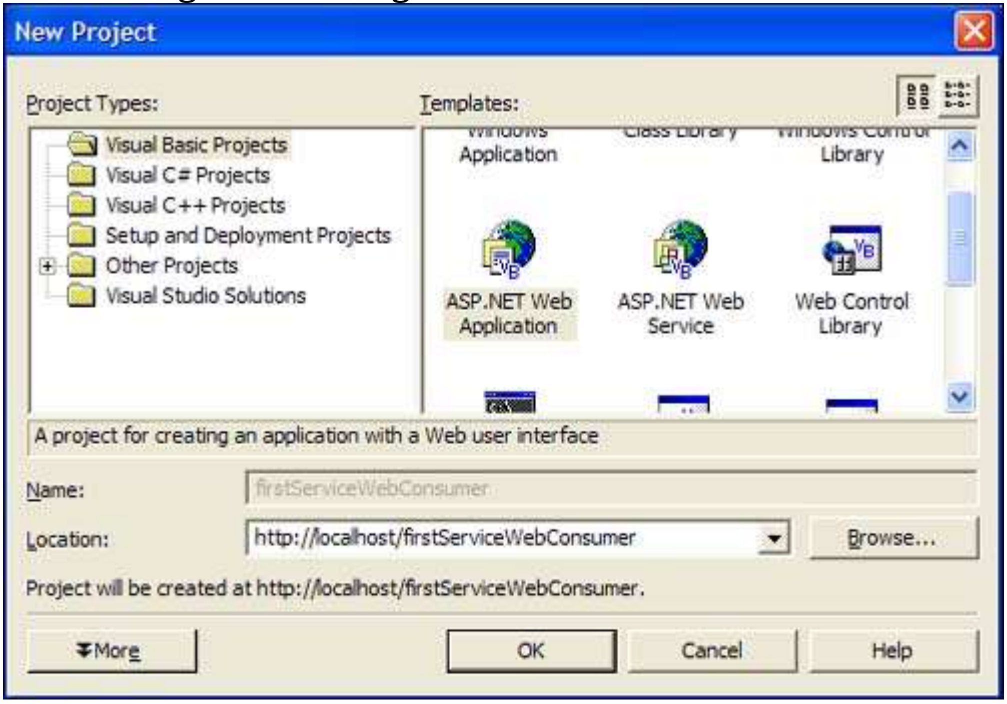
There are probably 50 ways to consume a Web service from any number of client applications. You can use Visual Basic 3, Visual Basic 4, Visual Basic 5, Visual Basic 6, Delphi, FLASH, C++, PowerBuilder—you name it. Because the implementation simply returns text in XML format, anything that can read text can consume a Web service. That's the whole point of a Web service: It can be created and consumed by anything.
The rest of the day focuses on four ways to consume a Web service, which I think are 99.9% of what you'll ever run into. You learn how to consume a Web service from

- An ASP.NET application
- A Windows Forms application
- A console application
- An HTML page from client-side VBScript/JavaScript

Consuming a Web Service from an ASP.NET Application

The idea behind a Web service is to be able to access application components from any type of application anywhere. To see how this works, you're going to create a new ASP.NET Web application and call the three methods of the firstService application you just finished creating.
To start, open a new instance of Visual Studio .NET and create a new ASP.NET Web application named firstServiceWebConsumer in either Visual Basic .NET or C#, whichever language you prefer (see next Figure). Click the OK button to create your new project.

Figure Creating the Web service Web consumer.



A project for creating an application with a Web user interface

Name: firstServiceWebConsumer

Location: http://localhost/firstServiceWebConsumer

Project will be created at http://localhost/firstServiceWebConsumer.

After the project is created, you must reference the Web service you created earlier. This is accomplished in a similar manner to adding a reference to an assembly.
If you right-click the References node on the Solution Explorer, you have two options: Add Reference and Add Web Reference, as next Figure demonstrates. Select Add Web Reference from the contextual menu.

Figure The Add Reference option in the Solution Explorer.

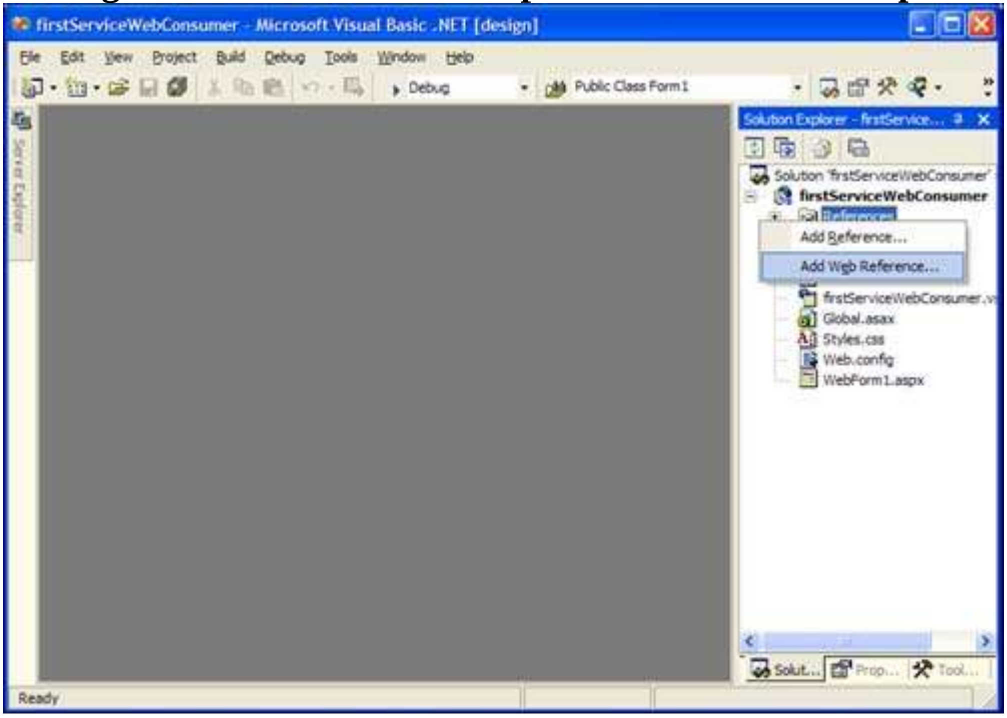
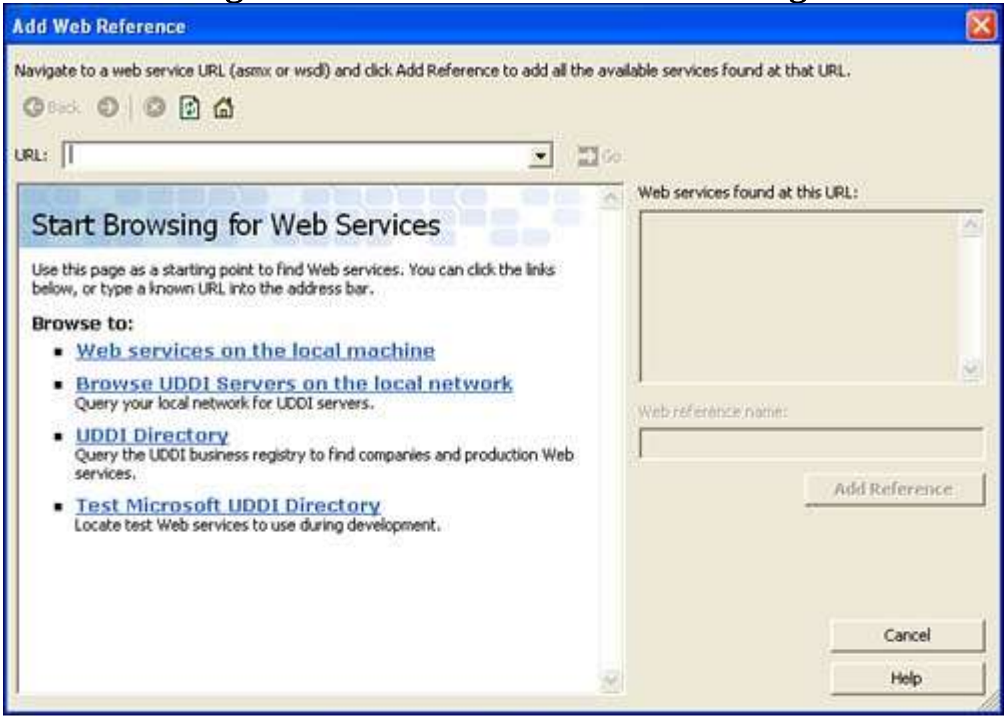

firstServiceWebConsumer - Microsoft Visual Basic .NET [design]
File Edit View Project Build Debug Tools Window Help
Debug
Public Class Form1
Solution Explorer - firstService...
Solution 'firstServiceWebConsumer'
firstServiceWebConsumer
References
Add Reference...
Add Web Reference...
firstServiceWebConsumer.v
Global.asax
Styles.css
Web.config
WebForm1.aspx
Ready

Figure The Add Web Reference dialog box.



Navigate to a web service URL (asmx or wsdl) and click Add Reference to add all the available services found at that URL.

Start Browsing for Web Services

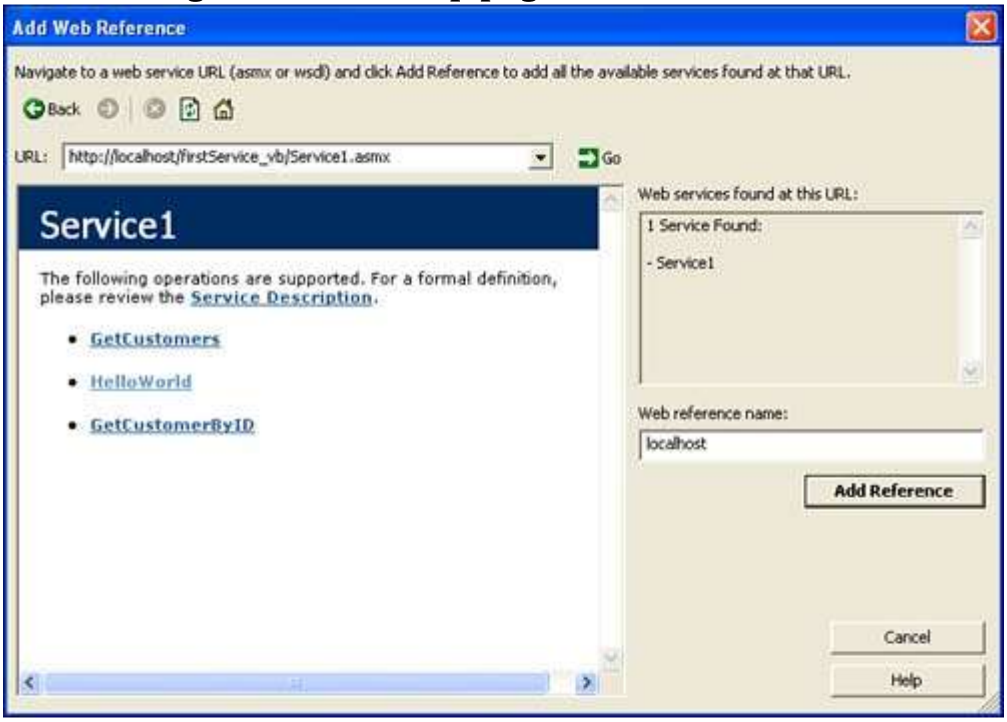
Use this page as a starting point to find Web services. You can click the links below, or type a known URL into the address bar.

Browse to:

- Web services on the local machine
- Browse UDDI Servers on the local network
  Query your local network for UDDI servers.
- UDDI Directory
  Query the UDDI business registry to find companies and production Web services.
- Test Microsoft UDDI Directory
  Locate test Web services to use during development.

The Add Web Reference dialog has several features. You can search the UDDI directory that was discussed earlier today. You can also search the Test Microsoft UDDI Directory. Microsoft has several Web services, such as the Knowledge Base search service, that you can test and implement in your own applications.
Because you don't want to do either of the available options, you must type the URL of the Web service you want to consume. In the Address box, type
http://localhost/fistService_vb/Service1.asmx
or
http://localhost/fistService_cs/Service1.asmx
depending on what you named your project earlier.
After you type in the address and press Enter, you should see something similar , which is the Service Help Page for the Web service named Service1.asmx.

Figure Service Help page for `Service1.asmx`.



Add Web Reference

Navigate to a web service URL (asmx or wsdl) and click Add Reference to add all the available services found at that URL.

URL: http://localhost/firstService_vb/Service1.asmx

Service1

The following operations are supported. For a formal definition, please review the Service Description.

- GetCustomers
- HelloWorld
- GetCustomerByID

Web services found at this URL:

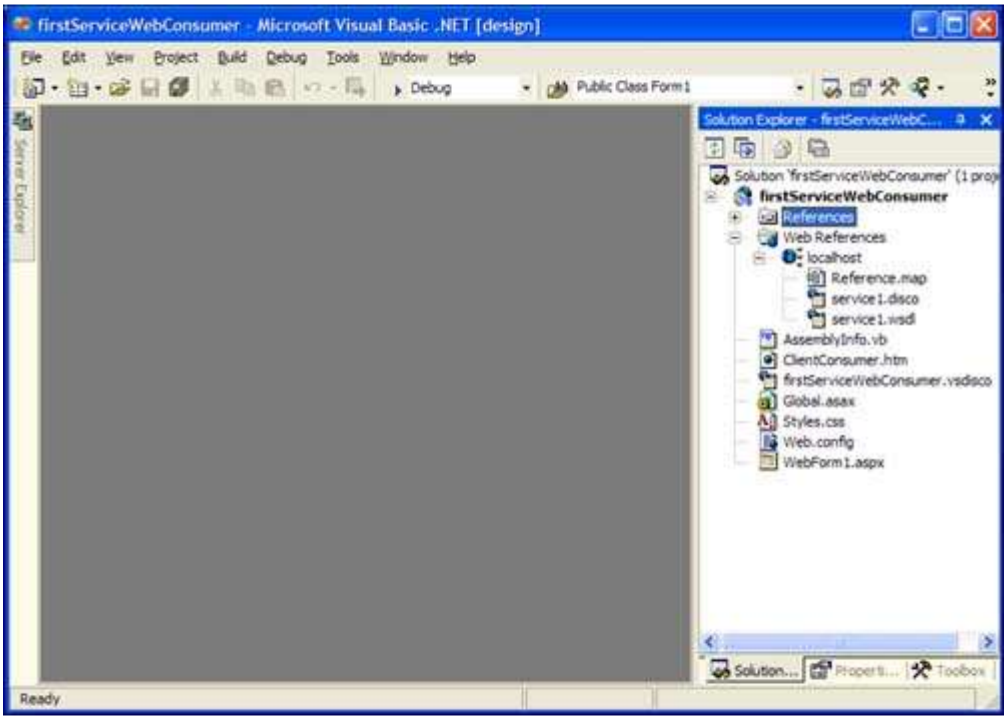
1 Service Found:

- Service1

Web reference name: localhost

From here, you have the same capabilities that you did when you ran the Web service from the project earlier. You can view the service descriptions, invoke the Web services, and inspect additional information that you might need about implementing the Web service. To add this reference to your project, click the Add Reference button. Your Solution Explorer should now look like:.

Figure Solution after adding a Web reference for `Service1.asmx`.



Tip

When you add the Web reference, the Web server name where the Web service is located shows up in the Solution Explorer under Web References. You can right-click on the name of the Web server (in this case, localhost) and change it to something more familiar or recognizable.

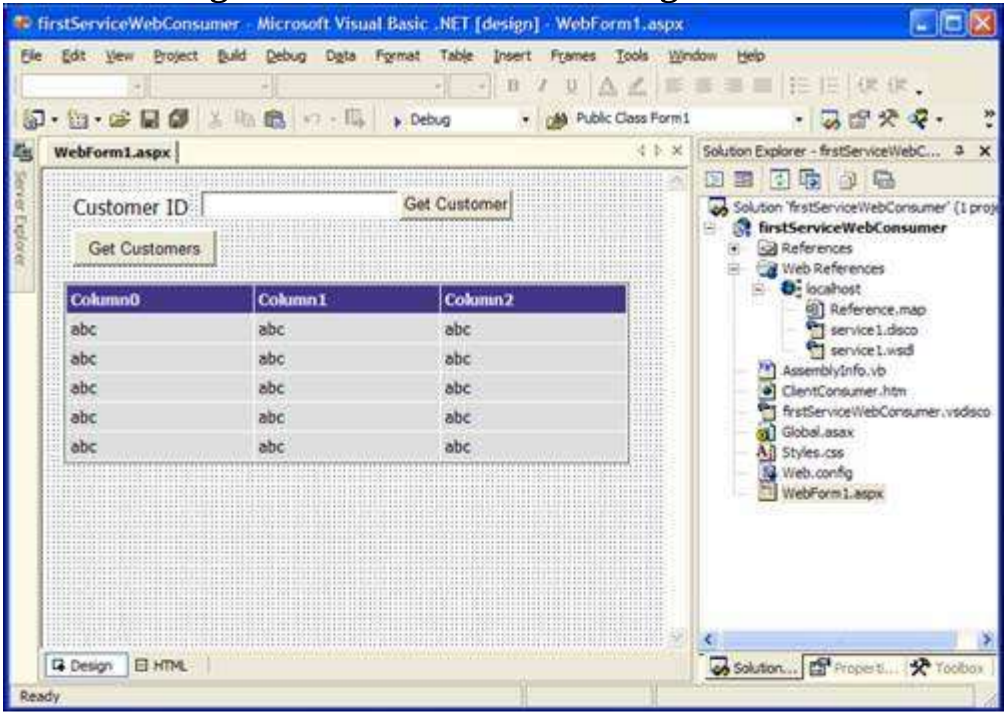
Now that you've added the Web reference, you can reference the methods in the Web service as you would any other method in a class. Remember that the WSDL file can be considered a type library for a Web service. In .NET, the term *proxy* is to refer to the metadata of an object, but not necessarily the object itself. When you add a reference to a Web service created in .NET, a WSDL proxy is added to your solution. The WSDL file contains all the type information that your application needs to invoke the Web service.

For this exercise, the Web service and consumer are on the same machine. This might or might not be true in real life. Either way, you should always use the actual DNS name or IP address of the Web service when you add it through the Add Web Reference dialog. That way, if you move the consumer to another machine, the Web Reference is still valid.

To access the methods in the newly added Web reference, follow these steps to prepare the default WebForm1.aspx page. The outcome should look familar.

1. Drag a Label control onto the form and set the Text property to Customer ID.
2. Drag a TextBox control to the right of the Label control and change its Name property to CustomerID.
3. Drag a CommandButton control to the form, change its Name property to GetCustomer and change the Text property to Get Customer.
4. Drag a CommandButton control to the form, change its Name property to GetCustomers, and change the Text property to Get Customers.
5. Drag a DataGrid control to the form. You can right-click the DataGrid and select AutoFormat from the contextual menu to select a nice color scheme.

Figure Webform1 after adding new controls.



Column0	Column1	Column2
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc

Now that you've added the controls and the form is all set, double-click the Get Customer button to get to the code-behind for the GetCustomer click event.

In Listing, you take the CustomerID information from the CustomerID TextBox and pass it to the GetCustomer method of the Web service.

C#

```
private void GetCustomer_Click(object sende,System.EventArgs e)
{
  localhost.Service1 ws = new localhost.Service1();

  DataGrid1.DataSource =
ws.GetCustomerByID(CustomerID.Text.Trim().ToString());

  DataGrid1.DataBind();
}
```

The code is amazingly simple. All you need to do is create an instance of the Web service reference, and then call the method on the reference like any other class. Because the DataGrid control natively reads a DataSet and data returned from an XML Web service is read as either XML or a DataSet, you can just set the DataSource property of the DataGrid to the return value of the Web service, and the DataGrid understands the return value. After you set the DataSource of the DataGrid, you can call the DataBind method of the DataGrid and the data appears in the grid.

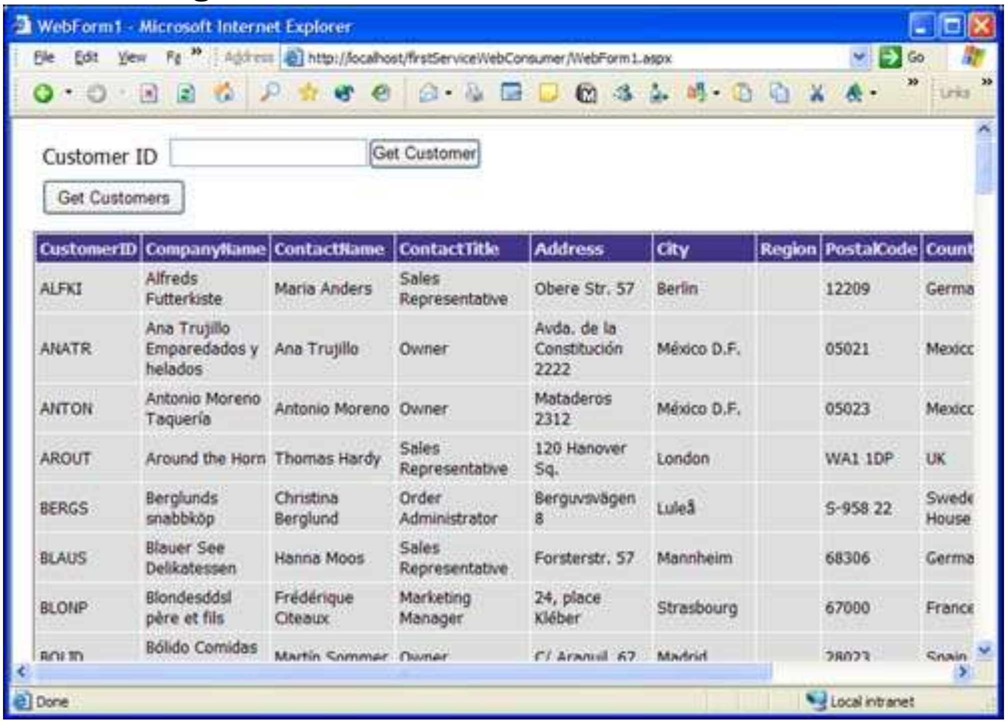
Listing the code you should add to the GetCustomers click event, which returns all the customers in the Customers table.

C#

```
private void GetCustomer_Click(object sende,System.EventArgs e)
{
  localhost.Service1 ws = new localhost.Service1();

  DataGrid1.DataSource = ws.GetCustomers();

  DataGrid1.DataBind();

}
```

Now that you've written the code for both Click events, you can press F5 to run the application.
When WebForm1 is in the browser, click the Get Customers button. Your results should look like next Figure.

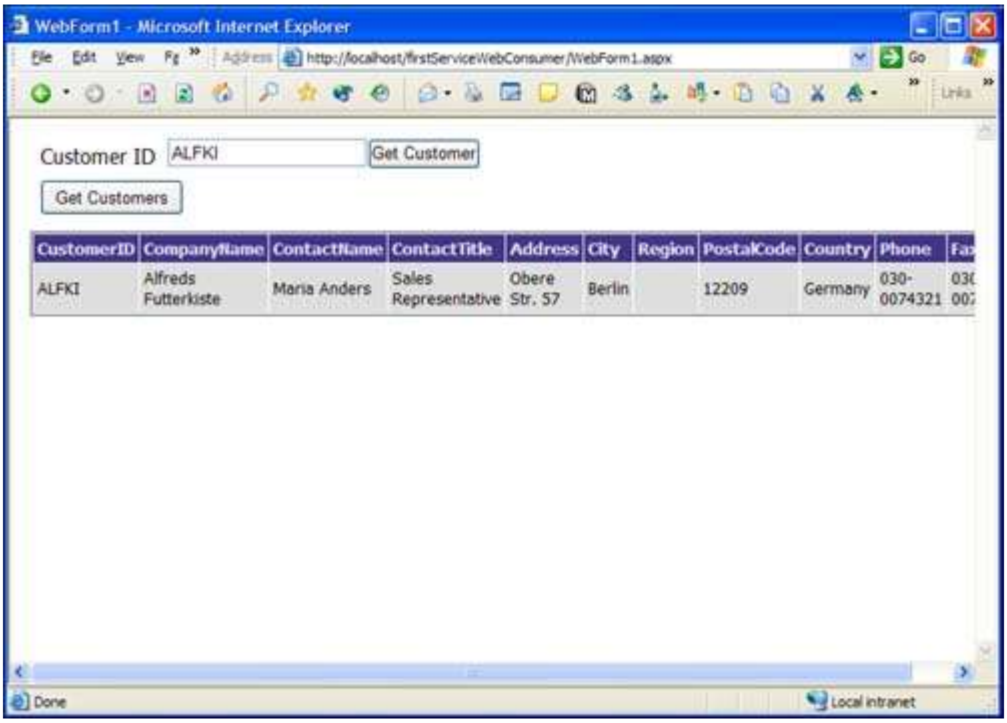
Figure Results of the `GetCustomers` method.


WebForm1 - Microsoft Internet Explorer
File Edit View Fa
Address http://localhost/firstServiceWebConsumer/WebForm1.aspx
Go
Links
Customer ID
Get Customer
Get Customers
Done
Local intranet

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Count
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germa
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexicc
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexicc
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1DP	UK
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Swede House
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim		68306	Germa
BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg		67000	France
BOLID	Bólido Comidas	Martín Sommer	Owner	C/ Araquil, 67	Madrid		28023	Spain

Next, enter the CustomerID **ALFKI** into the TextBox control and click Get Customer. You should see something similar to next Figure.

Figure Results of the `GetCustomer` method.


WebForm1 - Microsoft Internet Explorer
File Edit View Fa
Address http://localhost/firstServiceWebConsumer/WebForm1.aspx
Go
Links
Customer ID ALFKI
Get Customer
Get Customers
Done
Local intranet

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fa
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074321	030 00

Just like that, you've consumed data from a Web service. You've also seen how to pass a variable to a method in a Web service. It's no different than working with any other object, except that methods can be called on a remote server and data is retrieved with no special settings on your part. It's truly a powerful tool.
If you don't want to simply bind the data to a DataGrid, you can save the data to a file, load it into a stream, or bind it to other controls.

Consuming an XML Web Service from a Windows Form

Consuming a Web service from a Windows Forms application is identical to consuming a Web service from an ASP.NET Web application. You need to add a Web reference to the project, create an instance of the proxy class, and call the methods that you need to consume.

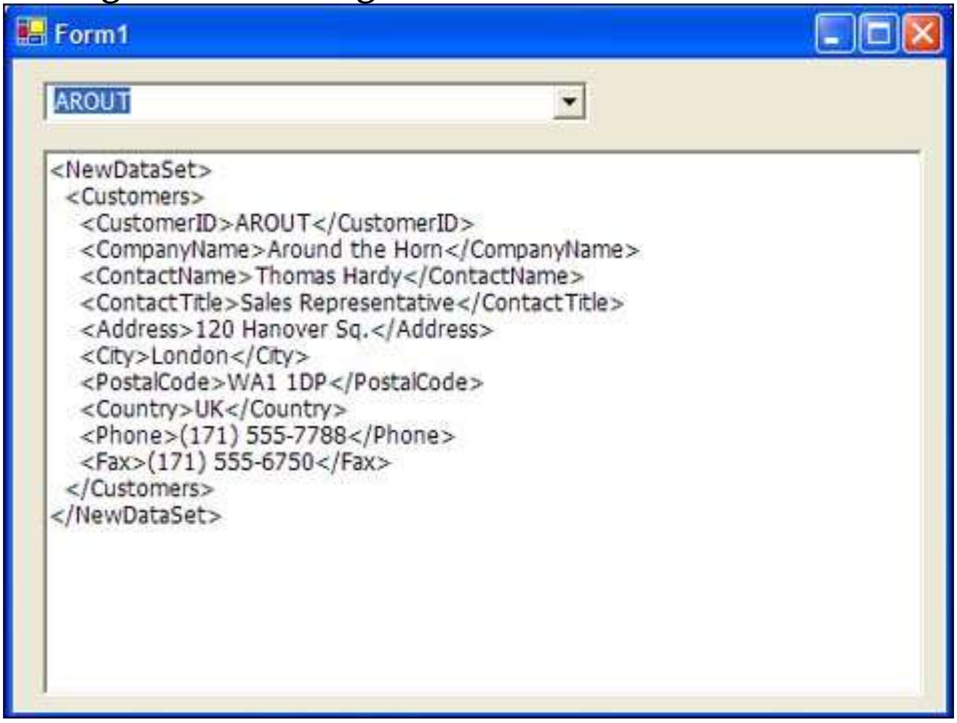
To test this, create a new Windows Forms application and name it firstServiceFormsConsumer_vb or firstServiceFormsConsumer_cs, depending on the language you're coding in.
When the project is loaded, add a ComboBox and a RichTextBox to the default form1. Change the Name property of the ComboBox to cbo, and change the Name property of the RichTextBox to rtb.
Next, add the Web reference to the project exactly as you did for the ASP.NET Web application. Right-click the References node in the Solution Explorer, select Add Web Reference from the contextual menu, and enter the URL to the ASMX file.
When that's done, you can write some code to consume the Web service. You're going to load the return data from the Web service into a DataSet, and then bind the DataSet to the ComboBox. On the SelectedIndexChanged event of the ComboBox, you'll call the GetCustomerByID Web service, and pass the Text property of the ComboBox to the method.

Listing shows the code you should add to both the Form_Load event and the SelectedIndexChanged event of the ComboBox.

```
private void Form1_Load(object sender, System.EventArgs e)
  {
    localhost.Service1  ws =  new localhost.Service1();
    DataSet  ds = ws.GetCustomers();
    cbo.DisplayMember = "CustomerID";
    cbo.DataSource = ds.Tables[0];
  }
private void comboBox1_SelectedIndexChanged(object sender,
System.EventArgs e)
  {
    localhost.Service1  ws =  new localhost.Service1();

    DataSet ds = ws.GetCustomerByID(cbo.Text);

    rtb.Text = ds.GetXml();
  }
```

If you run the application by pressing the F5 key, the Form_Load event calls the Web service and grabs all the customers. The databinding used in the form load is the same code you used two days ago when you wrote the DataAccess application. Because the return type from the Web service is a DataSet, you can just set the DataSet equal to the data coming back from the Web service. After the ComboBox is bound, the SelectedIndexChanged event fires and the GetCustomerID method of the Web service proxy is invoked each time you select another item. The GetXml method that you learned about yesterday loads the XML string into the RichTextBox control.

Figure Consuming the Web service from Windows Forms.



```
<NewDataSet>
  <Customers>
    <CustomerID>AROUT</CustomerID>
    <CompanyName>Around the Horn</CompanyName>
    <ContactName>Thomas Hardy</ContactName>
    <ContactTitle>Sales Representative</ContactTitle>
    <Address>120 Hanover Sq.</Address>
    <City>London</City>
    <PostalCode>WA1 1DP</PostalCode>
    <Country>UK</Country>
    <Phone>(171) 555-7788</Phone>
    <Fax>(171) 555-6750</Fax>
  </Customers>
</NewDataSet>
```

Each time you select another CustomerID from the ComboBox, the XML changes.

Because both methods from the Web service are returning a DataSet, you have the full power of the DataSet class at your fingertips. You can save the XML to a file, load the XML into an XMLDocument object, or use an XMLTextReader to read through the nodes. Everything you learned yesterday about working with XML can be applied to the results from an XML Web service.
If the return type is a string, integer, or even a serialized class, you can handle it accordingly. If a client is non-.NET, the data is just XML, so it can be treated like an XML file.