Beginners Guide to ASP.NET

CONTENTS

ASP .NET Introduction

Prerequisites

Before you continue you should have a basic understanding of the following:

- WWW, HTML and the basics of building Web pages
- Scripting languages like JavaScript or VBScript
- The basics of server side scripting

What is .NET Framework & what is ASP .NET?

.NET Framework is a very broad infrastructure for application development and deployment. ASP .NET is a part of the new .NET Framework. It is used to build web applications.

Here are some more points about .NET Framework

- The .NET Framework is the infrastructure for the new Microsoft .NET Platform.
- The .NET Framework is a common environment for building, deploying, and running Web applications and Web Services.
- The .NET Framework contains a common language runtime and common class libraries - like ADO .NET, ASP .NET and Windows Forms - to provide advanced standard services that can be integrated into a variety of computer systems.
- The .NET Framework provides a feature-rich application environment, simplified development and easy integration between a number of different development languages.
- The .NET Framework is language neutral. Currently it supports C++, C#, Visual Basic, and JScript (The Microsoft version of JavaScript).
- Microsoft's Visual Studio.NET is a common development environment for the new .NET Framework.

Dynamic Page in ASP .NET

This following code displays our example as an ASP .NET page:

```
<html>
<body bgcolor="yellow">
<center>
<h2>Hello DC Bombay</h2>
<p><%Response.Write(now())%></p>
</center>
</body>
</html>
```

If you want to try it yourself, save the code in a file called "dynapage.aspx" under any virtual directory and call it using any browser with the HTTP path.

ASP .NET - Server Controls

Limitations in Classic ASP

The listing below was copied from the previous chapter:

```
<html>
<body bgcolor="yellow">
<center>
<h2>Hello DC Bombay</h2>
<p><%Response.Write(now())%></p>
</center>
</body>
</html>
```

The code above illustrates a limitation in Classic ASP: The code block has to be placed where you want the output to appear. With Classic ASP it is impossible to separate executable code from the HTML itself. This makes the page difficult to read, and difficult to maintain.

ASP .NET - Server Controls

ASP .NET has solved the "spaghetti-code" problem described above with server controls.
Server controls are tags that are understood by the server.
There are three kinds of server controls:

- HTML Server Controls - Traditional HTML tags
- Web Server Controls - New ASP .NET tags
- Validation Server Controls - For input validation

HTML Server Controls

HTML server controls are HTML tags understood by the server. HTML elements in ASP .NET files are, by default, treated as text i.e they are not interpreted by server. To make these elements programmable, add a runat="server" attribute to the HTML element. This attribute indicates that the element should be treated as a server control. The id attribute is added to identify the server control. The id reference can be used to manipulate the server control at run time i.e. it is the name used to refer the control in code.

Note: All HTML server controls must be within a <form> tag with the runat="server" attribute. The runat="server" attribute indicates that the form should be processed on the server. It also indicates that the enclosed controls can be accessed by server scripts.

In the following example we declare an HtmlAnchor server control in an .aspx file. Then we manipulate the HRef attribute of the HtmlAnchor control in an event handler (an event handler is a subroutine that executes code for a given event). The Page_Load event is one of many events that ASP .NET understands:

```
<script runat="server">
Sub Page_Load
  link1.HRef="http://www.yahoo.com"
```

```
  End Sub
</script>
<html>
<body>
  <form runat="server">
    <a id="link1" runat="server">Visit Yahoo!</a>
  </form>
</body>
</html>
```

The HTML Server controls are generally used when you are migrating legacy ASP applications to ASP.NET application. Just adding the runat=server make the controls work as .NET controls.

Web Server Controls

Web server controls are special ASP .NET tags understood by the server.

Like HTML server controls, Web server controls are also created on the server and they require a runat="server" attribute to work. However, Web server controls do not necessarily map to any existing HTML elements and they may represent more complex elements.

The syntax for creating a Web server control is:

```
<asp:control_name id="some_id" runat="server" />
```

In the following example we declare a Button server control in an .aspx file. Then we create an event handler for the Click event which changes the text on the button:

```
<script runat="server">
  Sub submit(Source As Object, e As EventArgs)
    button1.Text="You clicked me!"
  End Sub
</script>
<html>
<body>
  <form runat="server">
    <asp:Button id="button1" Text="Click me!"
                           runat="server" OnClick="submit"/>
  </form>
</body>
</html>
```

The Web Server controls should be used with server side ASP.NET code as much as possible because they were created specifically for that purpose. They offer better consistency as compared to HTML Server Controls.

Validation Server Controls

Validation server controls is used to validate user-input. If the user-input does not pass validation, it will display an error message to the user.

Each validation control performs a specific type of validation (like validating against a specific value or a range of values).

By default, page validation is performed when a Button, ImageButton, or LinkButton control is clicked. You can prevent validation when a button control is clicked by setting the CausesValidation property to false.

The syntax for creating a Validation server control is:

```
<asp:control_name id="some_id" runat="server" />
```

In the following example we declare one TextBox control, one Button control, and one RangeValidator control in an .aspx file. If validation fails, the text "The value must be from 1 to 100!" will be displayed in the RangeValidator control:

```
<html>
<body>
<form runat="server">
  Enter a number from 1 to 100:
  <asp:TextBox id="tbox1" runat="server" />
   <br /><br />
  <asp:Button Text="Submit" runat="server" />
   <br />
  <asp:RangeValidator ControlToValidate="tbox1" MinimumValue="1"
MaximumValue="100" Type="Integer" EnableClientScript="false"
Text="The value must be from 1 to 100!" runat="server" />
</form>
</body>
</html>
```

Using HTML Controls and Server Controls

You can use server controls or HTML controls on a Web form. What' s the difference? Basically, server controls are a superset of HTML controls and offer the advantages shown below.

Feature	Server controls	HTML controls
Server events	Trigger control-specific events on the server.	Can trigger only page-level events on server (post-back)
State management	Data entered in a control is maintained across requests.	Data is not maintained; must be saved and restored using page-level scripts
Adaptation	Automatically detect browser and adapt display as appropriate.	No automatic adaptation; must detect browser in code or write for least common denominator
Properties	The .NET Framework provides a set of properties for each control. Properties allow you to change the control' s appearance and behavior within server-side code.	HTML attributes only

Server and HTML controls provide overlapping functionality. In general, it is easier to work with server controls. Table below lists the server controls and HTML controls by programming task.

Task	Server controls	HTML controls
Display text	Label, TextBox, Literal	Label, Text Field, Text Area, Password Field
Display tables	Table, DataGrid	Table
Select from list	DropDownList, ListBox, DataList, Repeater	List Box, Dropdown
Perform commands	Button, LinkButton, ImageButton	Button, Reset Button, Submit Button
Set Values	CheckBox, CheckBoxList, RadioButton, RadioButtonList	Checkbox, Radio Button
Display images	Image, ImageButton	Button, Reset Button, Submit Button
Navigation	Hyperlink	none (use <a> tags in text)
Group controls	Panel, Placeholder	Flow Layout, Grid Layout
Work with dates	Calendar	none
Display ads	AdRotator	none
Display horizontal rules	Literal	Horizontal Rule
Get filenames from client	none	File Field
Store data on page	(provided by state management)	Input Hidden
Validate data	RequiredFieldValidator, CompareValidator, RangeValidator, RegularExpressionValidator, CustomValidator,ValidationSummary	none (use page-level)

Text displayed in Labels and TextBoxes is arranged in a single block. To arrange text into rows and columns, you need to use one of the list or table controls described in Table 4-4. Use the ListBox, DropDownList, and Table controls for simple dynamic tables and lists. Use the DataGrid, DataList, and Repeater controls for complex tables and lists that contain other controls or are bound to data.

Control	Used To
ListBox	Display read-only text in a simple scrollable list format.
DropDownList	Display read-only text in a simple drop-down list format.
Table	Display text and controls in columns and rows. Table controls allow you to dynamically build tables in code using TableRows and TableCells

	collections.
DataGrid	Display text and controls in columns and rows using a template to control appearance. DataGrid controls have built-in formatting, sorting, and paging capabilities.
DataList	Display rows of text and controls using a template to control appearance. DataList controls have built-in formatting and selection capabilities.
Repeater	Display rows of other controls using a template to control appearance. Repeater controls do not include the built-in capabilities found in the DataGrid and DataList controls.

ASP .NET – Events

An Event Handler is a subroutine that executes code for a given event.

ASP .NET - Event Handlers

Look at the following code:

```
<%
lbl1.Text="The date and time is " & now()
%>
<html>
<body>
<form runat="server">
<h3><asp:label id="lbl1" runat="server" /></h3>
</form>
</body>
</html>
```

When will the code above be executed? The answer is: "You don't know..."

The Page_Load Event

The Page_Load event is one of many events that ASP .NET understands. The Page_Load event is triggered when a page loads, and ASP .NET will automatically call the subroutine Page_Load, and execute the code inside it:

```
<script runat="server">
  Sub Page_Load
    lbl1.Text="The date and time is " & now()
  End Sub
</script>
<html>
<body>
  <form runat="server">
    <h3><asp:label id="lbl1" runat="server" /></h3>
  </form>
</body>
</html>
```

Note: The Page_Load event contains no object references or event arguments!

The Page.IsPostBack Property

The Page_Load subroutine runs EVERY time the page is loaded. If you want to execute the code in the Page_Load subroutine only the FIRST time the page is loaded, you can use the Page.IsPostBack property. If the Page.IsPostBack property is false, the page is loaded for the first time, if it is true, the page is posted back to the server (i.e. from a button click on a form):

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
  lbl1.Text="The date and time is " & now()
end if
```

```
End Sub
Sub submit(s As Object, e As EventArgs)
lbl2.Text="Hello World!"
End Sub
</script>
<html>
<body>
<form runat="server">
<h3><asp:label id="lbl1" runat="server" /></h3>
<h3><asp:label id="lbl2" runat="server" /></h3>
<asp:button text="Submit" onclick="submit" runat="server" />
</form>
</body>
</html>
```

The example above will write the "The date and time is...." message only the first time the page is loaded. When a user clicks on the Submit button, the submit subroutine will write "Hello World!" to the second label, but the date and time in the first label will not change.

ASP .NET Web Forms

All server controls must appear within a <form> tag, and the <form> tag must contain the runat="server" attribute.

ASP .NET Web Forms

All server controls must appear within a <form> tag, and the <form> tag must contain the runat="server" attribute. The runat="server" attribute indicates that the form should be processed on the server. It also indicates that the enclosed controls can be accessed by server scripts:

```
<form runat="server">
...HTML + server controls
</form>
```

Note: The form is always submitted to the page itself. If you specify an action attribute, it is ignored. If you omit the method attribute, it will be set to method="post" by default. Also, if you do not specify the name and id attributes, they are automatically assigned by ASP.NET.

Note: An .aspx page can only contain ONE <form runat="server"> control!

If you select view source in an .aspx page containing a form with no name, method, action, or id attribute specified, you will see that ASP .NET has added these attributes to the form. It looks something like this:

```
<form name="_ctl0" method="post" action="page.aspx" id="_ctl0">
...some code
</form>
```

Submitting a Form

A form is most often submitted by clicking on a button. The Button server control in ASP .NET has the following format:

```
<asp:Button id="id" text="label" OnClick="sub" runat="server" />
```

The id attribute defines a unique name for the button and the text attribute assigns a label to the button. The onClick event handler specifies a named subroutine to execute.

In the following example we declare a Button control in an .aspx file. A button click runs a subroutine which changes the text on the button:

ASP .NET Maintaining the ViewState

You may save a lot of coding by maintaining the ViewState of the objects in your Web Form.

Maintaining the ViewState

When a form is submitted in classic ASP, all form values are cleared. Suppose you have submitted a form with a lot of information and the server comes back with an error. You will have to go back to the form and correct the information. You click the back button, and what happens.....ALL form values are CLEARED, and you will have to start all over again! The site did not maintain your ViewState.

When a form is submitted in ASP .NET, the form reappears in the browser window together with all form values. How come? This is because ASP .NET maintains your ViewState. The ViewState indicates the status of the page when submitted to the server. The status is defined through a hidden field placed on each page with a <form runat="server"> control. The source could look something like this:

```
<form name="_ctl0" method="post" action="page.aspx" id="_ctl0">
<input type="hidden" name="__VIEWSTATE"
value="dDwtNTI0ODU5MDE1Ozs+ZBCF2ryjMpeVgUrY2eTj79HNl4Q=" />
.....some code
</form>
```

Maintaining the ViewState is the default setting for ASP.NET Web Forms. If you want to NOT maintain the ViewState, include the directive <%@ Page EnableViewState="false" %> at the top of an .aspx page or add the attribute EnableViewState="false" to any control.

Look at the following .aspx file. It demonstrates the "old" way to do it. When you click on the submit button, the form value will disappear:

```
<html>
<body>
<form action="demo_classicasp.aspx" method="post">
Your name: <input type="text" name="fname" size="20">
<input type="submit" value="Submit">
</form>
<%
dim fname
fname=Request.Form("fname")
If fname<>"" Then
Response.Write("Hello " & fname & "!")
End If
%>
</body>
</html>
```

Here is the new ASP .NET way. When you click on the submit button, the form value will NOT disappear:

```
<script runat="server">
Sub submit(sender As Object, e As EventArgs)
```

```
lbl1.Text="Hello " & txt1.Text & "!"
End Sub
</script>
<html>
<body>
<form runat="server">
Your name: <asp:TextBox id="txt1" runat="server" />
<asp:Button OnClick="submit" Text="Submit" runat="server" />
<p><asp:Label id="lbl1" runat="server" /></p>
</form>
</body>
</html>
```

Database Connection

ADO .NET is a part of the .NET Framework and it is used to handle data access. With ADO .NET you can work with databases.

What is ADO .NET?

- ADO .NET is a part of the .NET Framework
- ADO .NET consists of a set of classes used to handle data access
- ADO .NET is entirely based on XML
- ADO .NET has, unlike ADO, no Recordset object

Create a Database Connection

We are going to use the Northwind database in our examples.

First, import the "System.Data.OleDb" namespace. We need this namespace to work with Microsoft Access and other OLE DB database providers. We will create the connection to the database in the Page_Load subroutine. We create a dbconn variable as a new OleDbConnection class with a connection string which identifies the OLE DB provider and the location of the database. Then we open the database connection:

```
<%@ Import Namespace="System.Data.OleDb" %>
<script runat="server">
sub Page_Load
dim dbconn
dbconn=New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
data source=" & server.mappath("northwind.mdb"))
dbconn.Open()
end sub
</script>
```

Note: The connection string must be a continuous string without a line break!

Create a Database Command

To specify the records to retrieve from the database, we will create a dbcomm variable as a new OleDbCommand class. The OleDbCommand class is for issuing SQL queries against database tables:

```
<%@ Import Namespace="System.Data.OleDb" %>
<script runat="server">
sub Page_Load
dim dbconn,sql,dbcomm
dbconn=New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
data source=" & server.mappath("northwind.mdb"))
dbconn.Open()
sql="SELECT * FROM customers"
dbcomm=New OleDbCommand(sql,dbconn)
end sub
</script>
```

Create a DataReader

The OleDbDataReader class is used to read a stream of records from a data source. A DataReader is created by calling the ExecuteReader method of the OleDbCommand object:

```
<%@ Import Namespace="System.Data.OleDb" %>
<script runat="server">
sub Page_Load
dim dbconn,sql,dbcomm,dbread
dbconn=New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
data source=" & server.mappath("northwind.mdb"))
dbconn.Open()
sql="SELECT * FROM customers"
dbcomm=New OleDbCommand(sql,dbconn)
dbread=dbcomm.ExecuteReader()
end sub
</script>
```

Bind to a Repeater Control

Then we bind the DataReader to a Repeater control:

```
<%@ Import Namespace="System.Data.OleDb" %>
<script runat="server">
sub Page_Load
dim dbconn,sql,dbcomm,dbread
dbconn=New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
data source=" & server.mappath("northwind.mdb"))
dbconn.Open()
sql="SELECT * FROM customers"
dbcomm=New OleDbCommand(sql,dbconn)
dbread=dbcomm.ExecuteReader()
customers.DataSource=dbread
customers.DataBind()
dbread.Close()
dbconn.Close()
end sub
</script>
<html>
<body>
<form runat="server">
<asp:Repeater id="customers" runat="server">
<HeaderTemplate>
<table border="1" width="100%">
<tr>
<th>Companyname</th>
<th>Contactname</th>
<th>Address</th>
<th>City</th>
</tr>
</HeaderTemplate>
<ItemTemplate>
<tr>
<td><%#Container.DataItem("companyname")%></td>
<td><%#Container.DataItem("contactname")%></td>
<td><%#Container.DataItem("address")%></td>
```

```
<td><%#Container.DataItem("city")%></td>
</tr>
</ItemTemplate>
<FooterTemplate>
</table>
</FooterTemplate>
</asp:Repeater>
</form>
</body>
</html>
```

Close the Database Connection

Always close both the DataReader and database connection after access to the database is no longer required:

```
dbread.Close()
dbconn.Close()
```

Web User Controls

Introduction

Web user controls combine one or more server or HTML controls on a Web User Control page, which can, in turn, be used on a Web form as a single control. User controls make it possible to create a single visual component that uses several controls to perform a specific task.

Once created, user controls can be used on Web forms throughout a project.

Creating and Using User Controls
There are five steps to creating and using a user control in a Web application:

1. Add a Web User Control page (.ascx) to your project.
2. Draw the visual interface of the control in the designer.
3. Write code to create the control' s properties, methods, and events.
4. Use the control on a Web form by dragging it from the Solution Explorer to the Web form where you want to include it.
5. Use the control from a Web form' s code by declaring the control at the module level and then using the control' s methods, properties, and events as needed within the Web form.

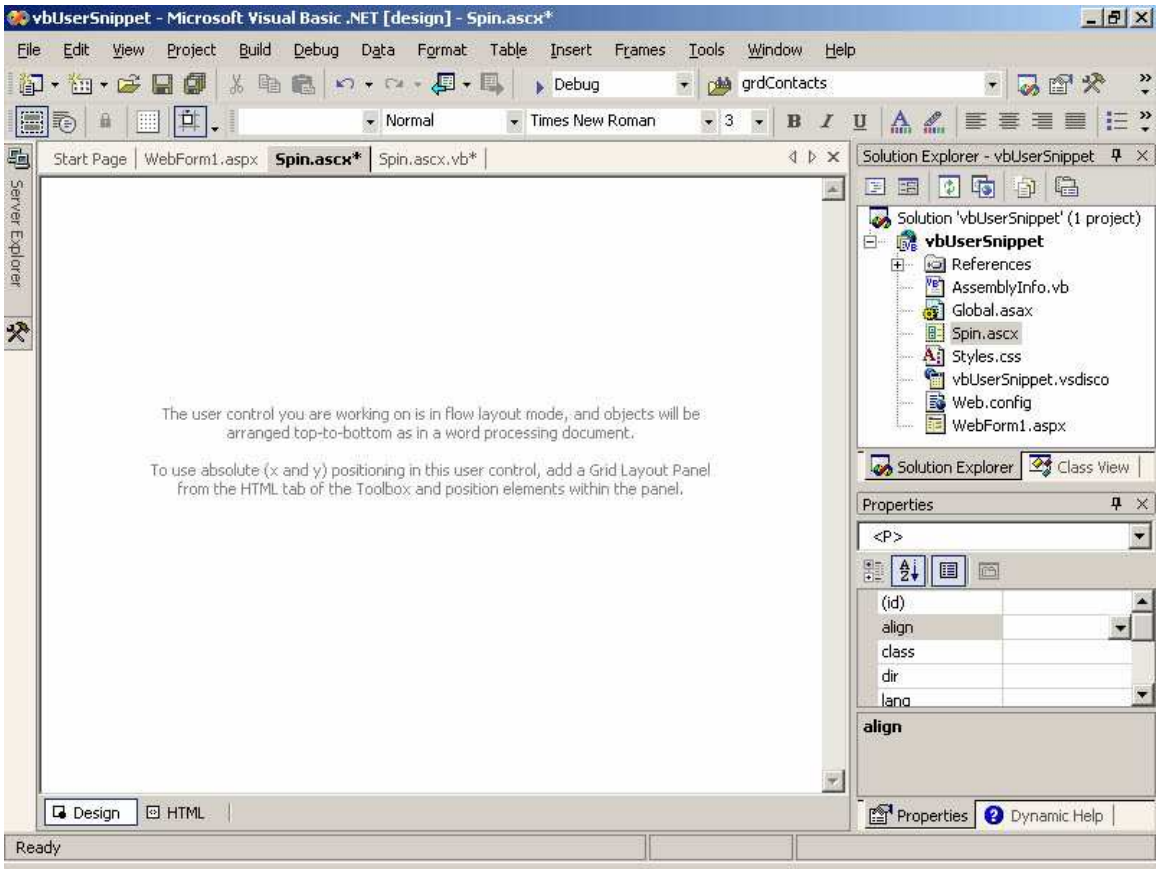
The following sections describe these steps in greater detail. The last section in this lesson describes special considerations you need to take into account when working with user controls.

Creating a User Control and Drawing Its Interface

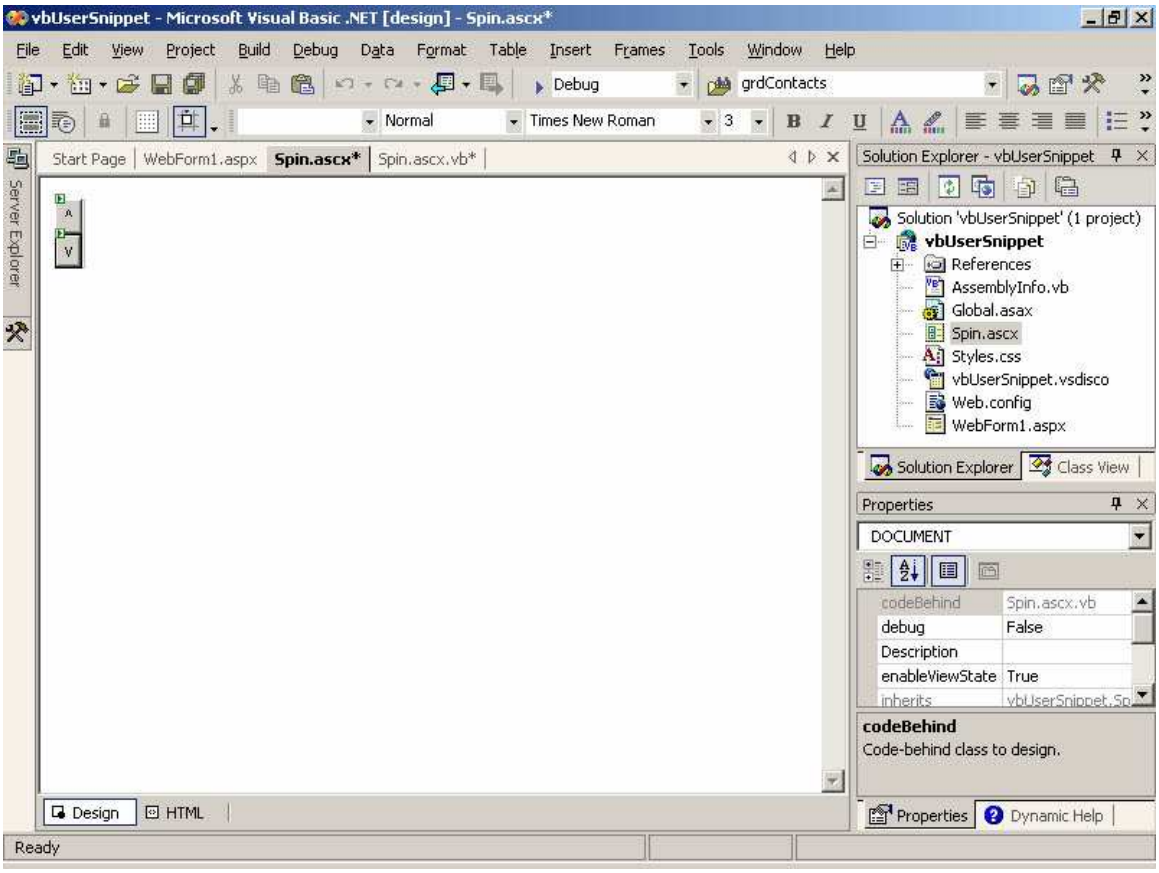
You create user controls out of other server and HTML controls in the Visual Studio .NET Toolbox. You do this by drawing the controls on a user control page, which is simply another file type in a Web application project, just like a Web form or an HTML page. User controls are identified by their .ascx filename extensions.

To create a user control and add it to a Web application, follow these steps:

- From the Project menu, choose Add Web User Control. Visual Studio .NET displays the Add New Item dialog box.
- Type the name of the user control in the Name text box and click OK. Visual Studio .NET creates a new, blank user control page and adds it to the project, as shown in below.



After you' ve added a user control page to your project, create the visual interface of the control by adding server or HTML controls to it. Figure below shows a simple user control created by placing two Button server controls on an HTML Grid Layout Panel control.



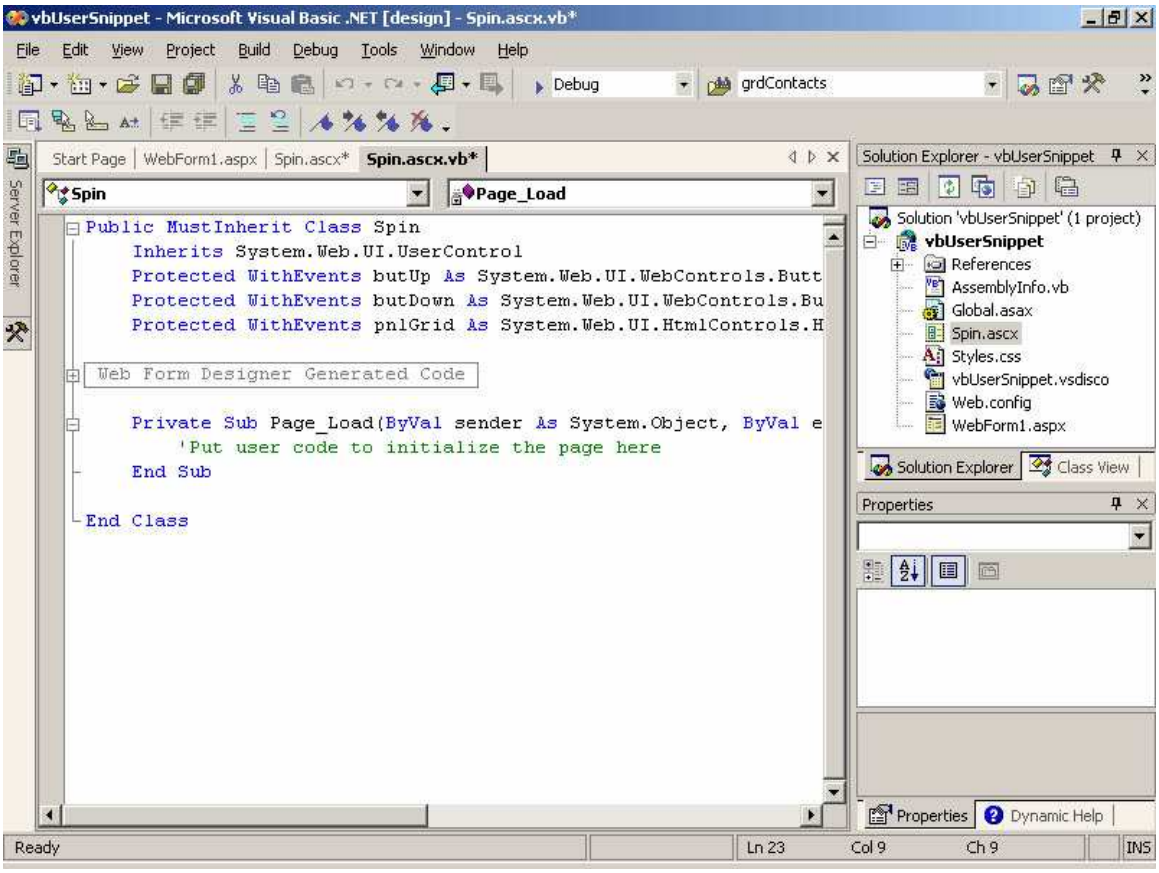
The control shown in figure above is the Web form' s equivalent of a Spin control. The control allows you to increment or decrement a value by clicking on the up or down arrow. This example is used in the sections that follow; the HTML for the user control is shown here for reference:

```
<DIV id="pnlGrid" runat="server" style="WIDTH: 20px; POSITION: relative;
HEIGHT: 48px" ms_positioning="GridLayout">
   <asp:Button id="butUp" Text="^" runat="server" />
   <asp:Button id="butDown" Text="v" runat="server" />
</DIV>
```

Writing the Control' s Properties, Methods, and Events

You manipulate the controls you add to the user control from the user control' s code module. That allows you to hide the inner workings of the control and expose the tasks the control performs as properties, methods, and events.

To edit the user control' s code module, simply double-click the user control. Visual Studio .NET automatically generates the code template shown in figure below when you create a user control.



The code in figure above is similar to the generated code for a Web form, with these notable differences:

- The user control' s class is declared as MustInherit in Visual Basic .NET or abstract in C#. The user control' s class can be used as a base class only for a derived class. In this case, the derived class is the user control created on the Web form.
- The user control' s class is based on the System.Web.UI.UserControl base class. This base class provides the base set of properties and methods you use to create the control and get the control' s design-time settings from its HTML attributes on the Web form.
- The user control' s Page_Load event occurs when the control is loaded by the Web form that contains it. The control' s Page_Load runs after the Web form' s Page_Load, which is important to remember when you work with user controls on a Web form.

To create properties and methods for the user control that you can use from a Web form, follow these steps:

1. Create the public property or method that you want to make available on the containing Web form.
2. Write code to respond to events that occur for the controls contained within the user control. These event procedures do the bulk of the work for the user control.
3. If the property or method needs to retain a setting between page displays, write code to save and restore settings from the control' s ViewState.

For example, the following code shows a Value property that returns the value of the Spin control created in the preceding section:

```
Public Property Value() As Integer
    Get
        ' Return the Value.
        Return ViewState("Value")
    End Get
    Set(ByVal Value As Integer)
        ' Set the Value.
        ViewState("Value") = Value
    End Set
End Property
```

The user changes the value of the Spin control by clicking the up or down buttons, so the following code increments or decrements the value, depending on which button the user clicks:

```
Private Sub butDown_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles butDown.Click
    ' Decrement the Value.
    Me.Value -= 1
End Sub

Private Sub butUp_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles butUp.Click
    ' Increment the Value.
    Me.Value += 1
End Sub
```

Adding the Control to a Web Form

User controls can be dragged directly from the Solution Explorer onto a Web form. When you add a user control to a Web form in this way, Visual Studio .NET generates a Register directive and HTML tags to create the control on the Web form.

For example, the following sample shows the HTML generated when you drag the Spin user control onto a Web form:

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="WebForm1.aspx.vb"
                                                Inherits="vbUserSnippet.WebForm1"%>
<%@ Register TagPrefix="uc1" TagName="Spin" Src="Spin.ascx" %>
<HTML>
   <body>
         <form id="Form1" method="post" runat="server">
               <uc1:Spin id="Spin1" runat="server"></uc1:Spin>
         </form>
   </body>
</HTML>
```

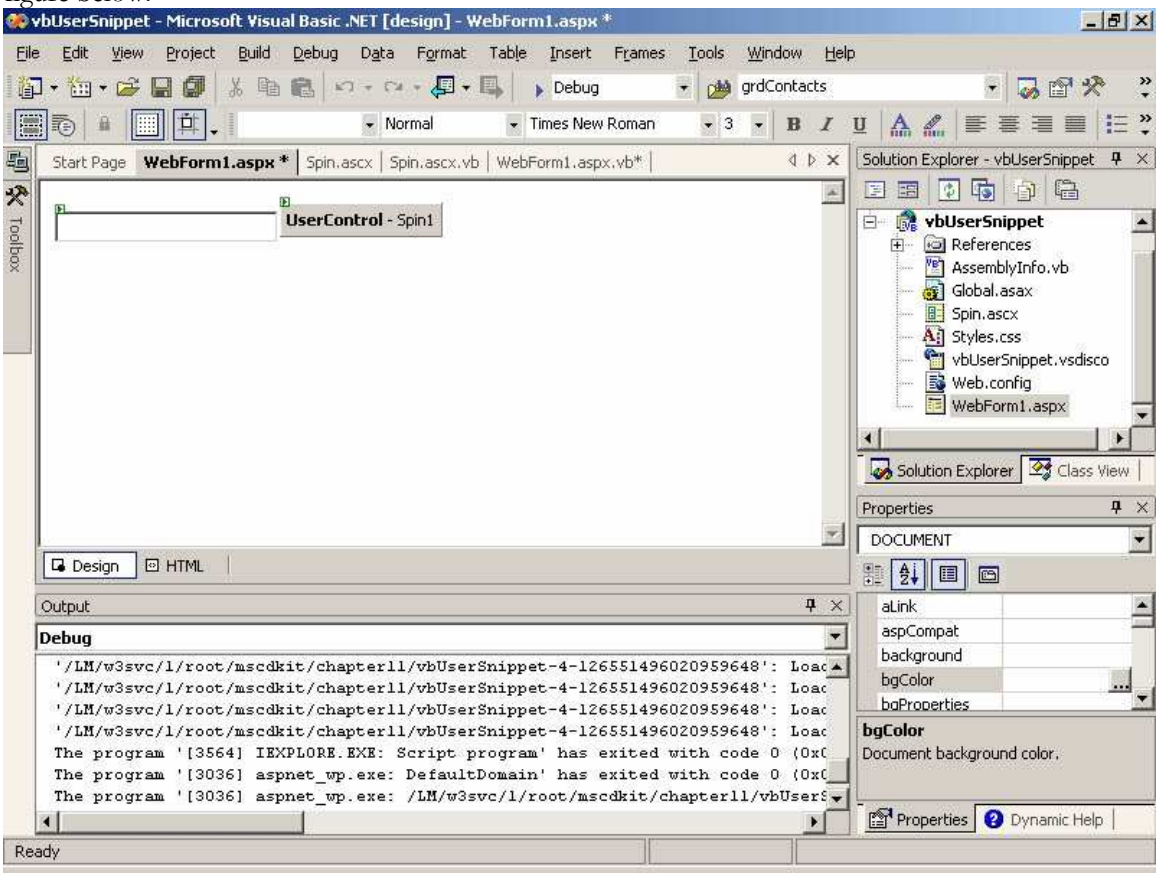
User controls can exist alongside and interact with other controls on a Web form. For example, the following HTML shows the Spin user control next to a TextBox control:

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="WebForm1.aspx.vb"
Inherits="vbUserSnippet.WebForm1"%>
```

```
<%@ Register TagPrefix="uc1" TagName="Spin" Src="Spin.ascx" %>
<HTML>
   <body>
         <form id="Form1" method="post" runat="server">
               <asp:TextBox id="TextBox1" runat="server"></asp:TextBox>
               <uc1:Spin id="Spin1" runat="server" Value="5"></uc1:Spin>
         </form>
   </body>
</HTML>
```

Notice that the user control includes an attribute for the Value property. When the Spin control loads at run time, ASP.NET will set the control' s properties based on the attributes in the HTML. The control' s attributes provide a way to set the control' s properties from HTML.

When you view the preceding HTML in Visual Studio .NET' s Design mode, it appears as shown in figure below.



As you' ll notice from figure above, the Web form designer does not display the user control as it will appear at run time. Instead, it shows a sort of generic control. This is a limitation of Visual Studio .NET —it can' t display user controls in Design mode.

Another thing you' ll notice is that the Spin control supports only flow layout on the Web form. That is a limitation of the control itself. To support grid layout and absolute positioning on the Web form, you have to add code to support the style attribute. You' ll see that technique later in this lesson.

Using the Control in Code

When you' ve created a user control and added it to a Web form, you can use it from the Web form's code module by following these steps:

1. Declare the user control at the module level. For example, the following line declares the Spin user control that was added to the Web form in the preceding section:

   ```
   Protected WithEvents Spin1 As Spin
   ```

2. Use the control' s properties, methods, and events as you would any other control. For example, the following event procedure displays the Spin control' s value in a text box:

   ```
   Private Sub Page_PreRender(ByVal sender As System.Object, _
      ByVal e As System.EventArgs) Handles MyBase.PreRender
       ' Display the Spin control's value.
       TextBox1.Text = Spin1.Value
   End Sub
   ```

One very important thing to notice here is that the preceding code uses the Web form' s Page_PreRender event procedure, not its Page_Load event procedure. If you use Page_Load, you' ll see only the Value set in the user control' s HTML Value attribute the first time you click the Spin control. (Try it!) That' s because the user control' s code doesn' t run until after the Web form' s Page_Load event has finished. Any changes that were saved in the control' s ViewState aren' t loaded until the control' s Page_Load has run.

Adding Events to the User Control

In addition to properties and methods, user controls can provide events that can respond to user actions on the Web form.

To add an event to a user control, follow these steps:

1. Declare the event within the user control' s code module. For example, the following code declares a Click event for the Spin user control:

   ```
   ' Declare event for Spin control.
   Public Event Click(ByVal sender As System.Object, ByVal e As
   System.EventArgs)
   ```

2. Create a method to raise the event. This step makes it easier for other classes to derive from this class, since they can override this method.

   ```
   ' Method to raise event.
   Protected Overridable Sub OnClick(ByVal e As EventArgs)
       RaiseEvent Click(Me, e)
   End Sub
   ```

3. Raise the event from within the user control' s code. For example, the following code raises the Click event whenever a user clicks the up or down buttons in the Spin user control:

   ```
   Private Sub butDown_Click(ByVal sender As System.Object, _
   ```

```
        ByVal e As System.EventArgs) Handles butDown.Click
         ' Decrement the Value.
         Me.Value -= 1
         ' Call the OnClick method.
         OnClick(e)
    End Sub

    Private Sub butUp_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butUp.Click
         ' Increment the Value.
         Me.Value += 1
         ' Call the OnClick method.
         OnClick(e)
    End Sub
```

To use the user control event from a Web form, include the user control on a Web form, as shown in the preceding two sections, and then write an event procedure that responds to the event. For example, the following code updates a text box when the user clicks the Spin control:

```
Private Sub Spin1_Click(ByVal sender As Object, _
   ByVal e As System.EventArgs) Handles Spin1.Click
    ' Update info from the user control's event handler.
    TextBox1.Text = Spin1.Value
End Sub
```