See This in MSDN Library

Creating Server Controls

Paul D. Sheriff
PDSA, Inc.

December 2003

Page Options

Summary: Server controls add functionality to Microsoft ASP.NET by bundling code and controls into a single unit. See how to create a simple server control in Microsoft Visual Studio .NET to display items from any database. (22 printed pages)

Microsoft® ASP.NET
Microsoft® Visual Studio® .NET

Download the source code for this article.

Contents

Creating a Web user control is simple. All you need to do is to copy and paste controls from an existing Web page onto a user control. However, since these controls are text files, with an optional code-behind file, the source files themselves must be placed into each project you wish to use them in. This creates multiple copies of the user control across multiple sites, which can become a maintenance nightmare.

In this article you will create a Web server control. These controls are a great way to create reusable UI and code that is stored in a DLL file. A DLL makes it easy to reuse this control in multiple Web sites and not have to worry about distributing the source code as you do with Web user controls. In addition, you can add a custom control to the toolbox.

Creating a Data-Aware DropDownList Control

One of the most common routines you will write when creating a Web application is filling a **DropDownList** control with data. To do this you will supply a **DataTextField** with the name of the field in a data source that you wish to use to fill the list portion of the control. You will need a **DataValueField**, which is used to retrieve the value from a field to use as the value field. You will use either a **DataAdapter** and a **DataSet**, or a **DataReader** to fill the control. Here is an example of the code you would write to fill a **DropDownList** control with data from the Categories table in the Microsoft® SQL Server™ database named Northwind.

```
' Visual Basic .NET
Private Sub CategoryLoad()
  Dim ds As DataSet
  Dim da As SqlDataAdapter
  Dim strSQL As String
  Dim strConn As String

  strSQL = "SELECT CategoryName, CategoryID "
  strSQL &= " FROM Categories"

  strConn = ConfigurationSettings. _
   AppSettings.Get("ConnectString")
  Try
    ds = New DataSet()
    da = New SqlDataAdapter(strSQL, strConn)

    da.Fill(ds)

    With ddlCat
      .DataTextField = "CategoryName"
      .DataValueField = "CategoryID"
      .DataSource = ds

      .DataBind()
    End With
  Catch ex As Exception
    Response.Write(ex.Message)

  End Try
End Sub

// C#
private void CategoryLoad()
{
   DataSet ds;
   SqlDataAdapter da;
   string strSQL;
   string strConn;

   strSQL = "SELECT CategoryName, CategoryID ";
   strSQL += " FROM Categories";

   strConn = ConfigurationSettings.
      AppSettings.Get("ConnectString");

   try
   {
      ds = new DataSet();
      da = new SqlDataAdapter(strSQL, strConn);

      da.Fill(ds);

      ddlCat.DataTextField = "CategoryName";
      ddlCat.DataValueField = "CategoryID";
      ddlCat.DataSource = ds;

      ddlCat.DataBind();
   }
   catch (Exception ex)
   {
      Response.Write(ex.Message);
   }
}
```

As you can see, this is quite a bit of code. While you could write your own class and a method to which you could pass in many of these parameters, a custom control would be very efficient in this case. When you are through with this article, you will be able replace all of the above code with code like the following:

```
' Visual Basic .NET
Private Sub CategoryLoad()
  With ddlCat
    .ConnectString = ConfigurationSettings. _
      AppSettings.Get("ConnectString")

    .DataTable = "Categories"
    .DataTextField = "CategoryName"
    .DataValueField = "CategoryID"

    .DataBind()
  End With
End Sub

// C#
private void CategoryLoad()
{
   ddlCat.ConnectString = ConfigurationSettings.
      AppSettings.Get("ConnectString");

   ddlCat.DataTable = "Categories";
   ddlCat.DataTextField = "CategoryName";
   ddlCat.DataValueField = "CategoryID";

   ddlCat.DataBind();
}
```

This is a big improvement, as you have cut down the amount of code you have to write by about 75 percent. In fact, you really will only need to set the **ConnectString** property and call the **DataBind** method, as the rest of the properties you can set at design time using the Properties window.

Creating the DataDropDownList Control

There are two different approaches to creating a Web server control. You may either add all the UI and functionality yourself by inheriting from **System.Web.UI.WebControls.WebControl**, or you may choose to inherit from an existing control and simply extend that control's functionality. It is much easier to extend an existing control than it is to build one from scratch. In this article you will learn to extend an existing control, such as the **DropDownList** control.

Your new control, the **DataDropDownList** control, will contain a standard **DropDownList** control, and will provide additional members as shown in Table 1. The **ConnectString** and **DataTable** properties are new to this control. The **DataBind** method is an override from the base **DropDownList** control, since you need to provide additional capabilities. All the rest of the properties, methods, and events are provided automatically, since we inherited from the **DropDownList** control.

Table 1. The new user control supplies these methods.

Member	Type	Description
ConnectString	Property (String)	Pass in a connection string to this property. This property is required to connect to the appropriate database that contains the table in the **DataTable** property.
DataTable	Property (String)	Set this property to the table name from the database that will be used to supply the data.
DataBind	Method	This method will override the **DropDownList** control's **DataBind** method.

The following sections walk you through creating your own **DataDropDownList** control.

Create the Web Server Control Project

To begin creating your **DataDropDownList** control, you need to create a new project in Microsoft® Visual Studio® .NET. Follow the steps below to accomplish this.

1. Start **Visual Studio .NET** and click **New Project**.
2. Click either **Visual Basic Projects** or **Visual C# Projects** in the **Project Types** list.
3. Select **Web Control Library** from the **Templates**.
4. Set the name to **DataDropDown**.
5. Set the location to any valid folder on your hard drive.
6. Click **OK** to create the new project.

Visual Studio .NET will create a new file named WebCustomControl1.vb or WebCustomControl1.cs that contains a template class with some attributes for your new custom control. You should see something like the following in your editor.

```
' Visual Basic .NET
Imports System.ComponentModel
Imports System.Web.UI

<DefaultProperty("Text"), _
  ToolboxData("<{0}:WebCustomControl1 _
  runat=server></{0}:WebCustomControl1>")> _
Public Class WebCustomControl1
    Inherits System.Web.UI.WebControls.WebControl

    Dim _text As String

    <Bindable(True), Category("Appearance"), _
     DefaultValue("")> Property [Text]() As String
        Get
            Return _text
        End Get

        Set(ByVal Value As String)
            _text = Value
        End Set
    End Property

    Protected Overrides Sub Render( _
     ByVal output As System.Web.UI.HtmlTextWriter)
        output.Write([Text])
    End Sub
End Class

// C#
using System;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.ComponentModel;

namespace DataDropDownCS
{
   /// <summary>
   /// Summary description for WebCustomControl1.
   /// </summary>
   [DefaultProperty("Text"),
      ToolboxData("<{0}:WebCustomControl1
      runat=server></{0}:WebCustomControl1>")]
 public class WebCustomControl1 :
  System.Web.UI.WebControls.WebControl
   {
      private string text;

      [Bindable(true),
         Category("Appearance"),
         DefaultValue("")]
      public string Text
      {
         get
         {
            return text;
         }

         set
         {
            text = value;
         }
      }
      /// <summary>
      /// Render this control to the output
      /// parameter specified.
      /// </summary>
      /// <param name="output"> The HTML writer to
      /// write out to </param>
      protected override void Render(
       HtmlTextWriter output)
      {
         output.Write(Text);
      }
   }
}
```

Unlike a user control, when you create your own Web server control, there is no design-time interface. Web server controls are built entirely from code. You will need to make a few changes to this standard template to create your own **DataDropDownList** control. Follow the steps below:

1. Change the **DefaultProperty("Text")** to **DefaultProperty("DataTable")**.
2. Change both the occurrences of **WebCustomControl1** within the **ToolboxData** attribute to **DataDropDownList**.
3. Change the **Class name** from **WebCustomControl1** to **DataDropDownList**.
4. Change the **file name** from **WebCustomControl1.vb** to **DataDropDownList.vb** or **DataDropDownList.cs**.
5. You now need to change where this control inherits from . It must inherit from **DropDownList** instead of **WebControl**.

   Visual Basic .NET

   ```
   Public Class DataDropDownList
     Inherits System.Web.UI.WebControls.DropDownList
   ```

   // C#

   ```
   public class DataDropDownList :
    System.Web.UI.WebControls.DropDownList
   ```

6. Delete the **Text** property declaration, including the line of code that declares the "_text" variable.
7. Finally you will change the **Render** event to call the **Render** method of the base **DropDownList** control.

When you are done, the code should look like the following:

```
' Visual Basic .NET
Imports System.ComponentModel
Imports System.Web.UI

<DefaultProperty("DataTable"), _
 ToolboxData("<{0}:DataDropDownList
  runat=server></{0}:DataDropDownList>")> _
 Public Class DataDropDownList
  Inherits System.Web.UI.WebControls.DropDownList

  Protected Overrides Sub Render( _
   ByVal output As System.Web.UI.HtmlTextWriter)
    MyBase.Render(output)
  End Sub
End Class

// C#
using System;
using System.Web.UI;
using System.Web.UI.WebControls;
```

```
using System.ComponentModel;

namespace DataDropDown
{
  /// <summary>
  /// Summary description for DataDrownDownList.
  /// </summary>
  [DefaultProperty("DataTable"),
  ToolboxData("<{0}:DataDropDownList
   runat=server></{0}:DataDropDownList>")]
  public class DataDropDownList :
    System.Web.UI.WebControls.DropDownList
  {

    /// <summary>
    /// Render this control to the output
    /// parameter specified.
    /// </summary>
    /// <param name="output"> The HTML writer to
    /// write out to </param>
    protected override void Render(HtmlTextWriter output)
    {
      base.Render(output);
    }
  }
}
```

Add a ConnectString Property

Since you want this new control to be able to read data from a data source, you must supply a connection string so it can connect to that data source. Add the following code to your **DataDropDownList** Class.

```
' Visual Basic .NET
Private mstrConnect As String = ""

<Bindable(True), Category("Data"), DefaultValue("")> _
Property ConnectString() As String
  Get
    Return mstrConnect
  End Get

  Set(ByVal Value As String)
    mstrConnect = Value
  End Set
End Property

// C#
private string mstrConnectString;

[Bindable(true),
Category("Data"),
DefaultValue("")]
public string ConnectString
{
   get   {return mstrConnectString;}
   set   {mstrConnectString = value;}
}
```

Notice the use of the **Bindable(True)**, **Category("Data")**, and **DefaultValue("")** attributes. These attributes inform the Visual Studio .NET environment how to display a property within the Property Window.

Add a DataTable Property

Since you need a table name to read the data from, you need to add a new property to your control into which you can place the table name. Add the following code to your control to add a **DataTable** property.

```
' Visual Basic .NET
Private mstrDataTable As String = ""

<Bindable(True), Category("Data"), DefaultValue("")> _
Public Property DataTable() As String
  Get
    Return mstrDataTable
  End Get
  Set(ByVal Value As String)
    mstrDataTable = Value
  End Set
End Property

// C#
private string mstrDataTable;

[Bindable(true),
Category("Data"),
DefaultValue("")]
public string DataTable
{
  get {return mstrDataTable;}
  set {mstrDataTable = value;}
}
```

Override the DataBind Event

Finally you must override the **DataBind** event so it can build the SQL statement, and use the connect string to hook up to the database and build a dataset to load the **DropDownList** control. Add the code to your server control that is shown below.

```
' Visual Basic .NET
Public Overrides Sub DataBind()
  Dim strSQL As String
  Dim strConn As String

  Try
    ' Build SQL String
    strSQL = String.Format("SELECT {0}, {1} FROM {2}", _
     MyBase.DataTextField, MyBase.DataValueField, _
     mstrDataTable)

    ' Fill in DataSource
    MyBase.DataSource = _
     Me.GetDataSet(strSQL, mstrConnect)

    ' Bind the Data
    MyBase.DataBind()

  Catch ex As Exception
    Throw ex

  End Try
End Sub

// C#
public override void DataBind()
{
   string strSQL;

   try
   {
      // Build SQL String
      strSQL = String.Format(
         "SELECT {0}, {1} FROM {2}",
         base.DataTextField, base.DataValueField,
         mstrDataTable);
      // Fill in DataSource
      base.DataSource =
         this.GetDataSet(strSQL, mstrConnectString);

      // Bind the Data
      base.DataBind();
   }
   catch (Exception ex)
   {
      throw ex;
   }
}
```

In the above code, you are building the SELECT statement dynamically using the **DataTextField**, **DataValueField** and the **DataTable** properties. Using the properties in this way makes this control very flexible. If you wish to use stored procedures to load this control, you would need to add an additional property to this control to pass in the stored procedure name to use to load this **DropDown** list. In that case, you would not build the SELECT in this routine, but instead use the stored procedure name.

The GetDataSet Method

The **DataBind** method calls a method named **GetDataSet**. The **GetDataSet** method is responsible for building a **DataSet** from the supplied SQL statement, and a connection string, and returning this **DataSet** back to the **DataBind** method. Add the code shown below to your server control.

```
' Visual Basic .NET
Private Function GetDataSet(ByVal SQL As String, _
ByVal ConnectString As String) As DataSet
  Dim ds As DataSet
  Dim da As SqlDataAdapter

  Try
    ds = New DataSet()
    da = New SqlDataAdapter(SQL, ConnectString)

    da.Fill(ds)

    Return ds

  Catch ex As Exception
    Throw ex

  End Try
End Function

// C#
private DataSet GetDataSet(string SQL,
 string ConnectString)
{
   DataSet ds;
   SqlDataAdapter da;

   try
   {
      ds = new DataSet();
      da = new SqlDataAdapter(SQL, ConnectString);

      da.Fill(ds);

      return ds;
   }
   catch (Exception ex)
   {
      throw ex;

   }
}
```

Building the Control

After you have completed adding all of the above properties and methods, you may now build your project to make sure you have no errors.

- Click **Build** | **Build Solution** from the **Visual Studio .NET** menu.

This will create a DLL named DataDropDown.dll in the Bin folder underneath the location where you created this project. Remember where the \Bin folder is located, as you will need to select the DLL from the test project that you will now create.

Testing Your Server Control

To test out your new Web server control, you will need to create a new Microsoft® ASP.NET Web application project. You will then be able to add your new server control to the Toolbox.

1. In the **Solution Explorer Window**, click the Solution, **DataDropDown**.
2. Right-click the solution and select **Add** | **New Project** from the context menu.
3. Click **ASP.NET Web Application** in the list of templates.
4. Set the name of the project to **ServerControl**.
5. Right-click on this new project and select **Set as Startup Project** from the context menu.
6. Right-click the toolbox and click **Customize Toolbox** in the context menu.
7. Click the **.NET Framework Components** tab.
8. Click **Browse**.
9. Navigate to the **ServerControl\Bin** folder and select the **DataDropDown.dll** file.
10. Click **Open**. You should now see the **DataDropDownList** control appear in the toolbox.
11. Double-click this new control, and it should draw the new control on the Web page.

You are now ready to fill in the design-time properties on this new **DataDropDownList** control. Fill in the properties of the **DataDropDownList** control with the values shown in Table 2.

Table 2. Set these properties to make your control display data from the Categories table in the Northwind database.

Property	Value
ConnectString	Server=(local);Database=Northwind;uid=sa;pwd=sa (or fill in whatever is appropriate for your database engine and database name)
DataTable	Categories
DataTextField	CategoryName
DataValueField	CategoryID
(ID)	ddlCat

Double-click anywhere on the WebForm1.aspx page to display the code-behind window. Write the following code into the Page_Load procedure.

```
' Visual Basic .NET
Private Sub Page_Load(ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles MyBase.Load
  ddlCat.DataBind()
End Sub

// C#
private void Page_Load(object sender, System.EventArgs e)
{
  ddlCat.DataBind();
}
```

Try it Out

You are now ready to run the project and test out your **DataDropDownList** control.

- Press **F5** to run the project.

If you did everything correctly, you should see a drop-down list of category names.

Custom Exception Handling

In your new custom control, you must fill in the **Connection String**, **DataTable**, **DataTextField**, and **DataValueField** properties. Since these are all required, you should check to see if any are blank prior to attempting to create the **DataSet** and binding to the **DataSet**. If any are blank, you should throw an exception with the names of the properties that are missing. You will add the **Check** method (see below) to your server control class. You will then call it from the **DataBind** method.

The Check Method

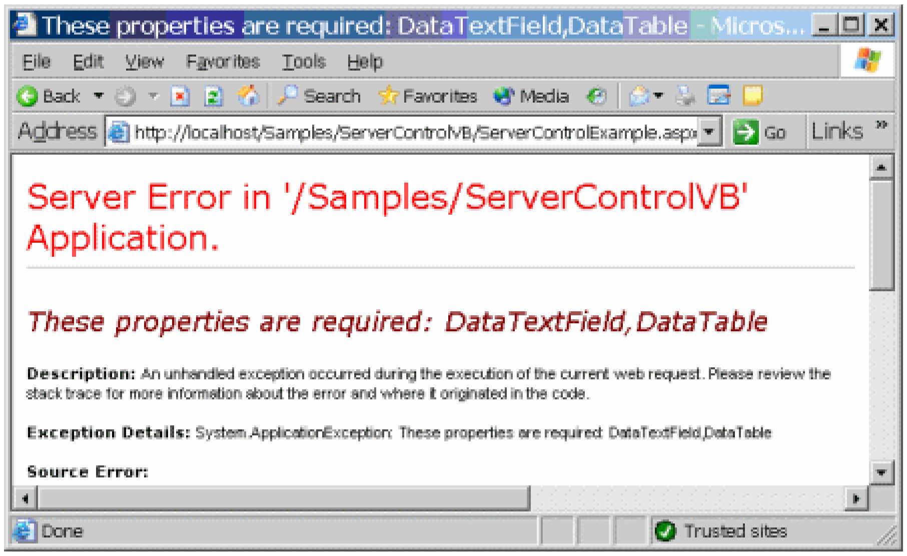
This method, named **Check**, is used to ensure that all of the properties are filled in prior to attempting to build the SQL statement and submit it to the back-end database.

```
' Visual Basic .NET
Private Sub Check()
  Dim strProp As String

  ' Check to see if all values
  ' are filled in correctly
  If DataTextField.Trim() = "" Then
    strProp &= "DataTextField"
  End If
  If DataValueField.Trim() = "" Then
    strProp &= ", DataValueField"
  End If
  If mstrDataTable.Trim() = "" Then
    strProp &= ",DataTable"
  End If
  If mstrConnectString.Trim() = "" Then
    strProp &= ",ConnectString"
  End If

  If strProp <> "" Then
    strProp = "These properties are required: " _
     & strProp
    Throw New ApplicationException(strProp)
  End If
End Sub

// C#
private void Check()
{
  string strProp = "";

  // Check to see if all values
  // are filled in correctly
  if (DataTextField.Trim() == "")
    strProp += "DataTextField";
  if (DataValueField.Trim() == "")
    strProp += ", DataValueField";
  if (mstrDataTable.Trim() == "")
    strProp += ",DataTable";
  if (mstrConnectString.Trim() == "")
    strProp += ",ConnectString";
  if (strProp != "")
  {
    strProp = "These properties are required: "
      + strProp;
    throw new ApplicationException(strProp);
  }
}
```

Now change the **DataBind** method to call the **Check** method. Add the lines of code shown in bold in the below listing.

```
' Visual Basic .NET
Public Overrides Sub DataBind()
  Dim strSQL As String
  Dim strConn As String

  Try
    Check()

    ' Build SQL String
    strSQL = String.Format("SELECT {0}, {1} FROM {2}", _
     MyBase.DataTextField, MyBase.DataValueField, _
     mstrDataTable)

   << CODE OMITTED >>
End Sub

// C#
public override void DataBind()
{
   string strSQL;

   try
   {
        Check();

      // Build SQL String
      strSQL = String.Format(
         "SELECT {0}, {1} FROM {2}",
         base.DataTextField, base.DataValueField,
         mstrDataTable);

        << CODE OMITTED >>
}
```

If you now go back and erase the values from the **DataTextField** and **DataValueField** properties in your control on your Web page, you should see an error that looks like the following:


These properties are required: DataTextField,DataTable - Micros...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/Samples/ServerControlVB/ServerControlExample.aspx Go Links
Done Trusted sites

Server Error in '/Samples/ServerControlVB' Application.

These properties are required: DataTextField, DataTable

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.ApplicationException: These properties are required: DataTextField,DataTable

Source Error:

Figure 1. Giving good error messages really helps the users of your Web controls.

Customizing Your Web Custom Control

To finish your custom control, you will probably want to supply a custom bitmap for the toolbox. In addition, the TagPrefix for these custom controls is currently set to "cc1," and you might wish to supply a better prefix for all of your own custom controls.

Changing the Toolbox Bitmap

To change the toolbox bitmap. you will first have to create a bitmap. There are a few requirements that must be met to change the bitmap that appears in the toolbox. First the bitmap must be a 16x16 pixel, 16-bit color bitmap, saved as a .BMP file. The bitmap must have the same file name as your control's class name. Since the name of your custom control is **DataDropDownList**, you will need to save your image as DataDropDownList.bmp. You will need to change the **Build Action** property of the bitmap control to **Embedded Resource**. This informs Visual Studio to compile the bitmap into the project. Follow the steps below to create a bitmap for your control:

1. Click **Project** | **Add New Item** from the **Visual Studio .NET menu**.
2. Select **Bitmap File** from the list of templates.
3. Set the name to **DataDropDownList**.
4. Click the new bitmap and set the **Build Action** property to **Embedded Resource**.
5. Draw anything you want.
6. Build the Project.
7. Switch back to the **ServerControl** project, and open the WebForm1.aspx page in design mode.
8. Right-click on the **DataDropDownList** control in the toolbox and click **Delete** in the context menu.
9. Re-add the **DataDropDownList** control back to the toolbox to see the new icon.

Modifying the TagPrefix

If you were to look at the HTML view in your WebForm1.aspx page, you would see a Register tag at the top. This tag informs the .ASPX page where the definition for this control comes from. If you look at this directive right now, you should see something that looks like the following:

```
<%@ Register TagPrefix="cc1" Namespace="WebControlSample"
```

```
 Assembly="WebControlSample" %>
```

If you then look further down within the <FORM> HTML tag, you will see the definition for your **DataDropDownList** control. You will see the "cc1" prefix that matches the TagPrefix in the Register directive.

```
<cc1:DataDropDownList id="DataDropDownList1"
 runat="server" ConnectString="Server=(local);
 Database=Northwind;uid=sa;pwd=sa" DataTable="Categories"
 DataTextField="CategoryName" DataValueField="CategoryID">
</cc1:DataDropDownList>
```

You will most likely wish to change this prefix as "cc1" is not very descriptive. To accomplish this, you will need to follow the steps below.

1. Double click on the **AssemblyInfo.vb** file within the WebControlSample project.
2. Add an **Imports** or a **using** statement at the top of the file to bring in the System.Web.UI namespace.

   ```
   ' Visual Basic .NET
   Imports System.Web.UI

   // C#
   using System.Web.UI;
   ```

3. You should then move down to where the other <Assembly:> tags are located and add the following tag:

   ```
   ' Visual Basic .NET
   <Assembly: TagPrefix("DataDropDown", "DataDD")>

   // C#
   [assembly: TagPrefix("DataDropDown", "DataDD")]
   ```

4. Rebuild your **ServerControl** solution.

You must now delete the control from the toolbox and re-add the control to the toolbox to have it refresh the **TagPrefix** attribute. You will also need to delete the control from your Web page, and you will most likely have to manually delete the old Register directive from the HTML view of the WebForm1.aspx file.

After adding the control back to your Web page, you can now view the HTML and see the new **TagPrefix** of "DataDD."

```
<%@ Register TagPrefix="DataDD" Namespace="DataDropDown"
 Assembly="DataDropDown" %>
```

You will also see the new prefix in the declaration of your custom control.

```
<DataDD:DataDropDownList id="ddlCat"
runat="server"></DataDD:DataDropDownList>
```

Of course when you delete the control, all of the design-time properties will be deleted as well. You will need to reset the design-time properties (**DataTable**, **ConnectString**, **DataTextField**, and **DataValueField**), or write the code as shown at the beginning of this article.

Conclusion

Creating a Web server control is very different from creating a Web user control. You will end up writing a lot less code in the UI layer when you employ a server control, as you can take advantage of inheritance. Since you will be creating a DLL, this will also ensure that no one can modify your source code. With a user control, anyone can view and see the source code. Another advantage of a server control over a user control is you only need one copy of the DLL on a single server. This one DLL can service all Web sites. With user controls, you would have to copy the user control from site to site. This makes maintenance of these user controls very difficult. With the server control, you only need to modify the control in one location.

About the Author

Paul D. Sheriff is the President of PDSA, Inc. (http://www.pdsa.com), a Microsoft Partner in Southern California. Paul is the Microsoft Regional Director for Southern California, and has four books on .NET. The first book is entitled *ASP.NET Developer's Jumpstart* from Addison-Wesley and is co-written with Ken Getz. His other three books are eBooks and can be purchased directly from the PDSA Web site. You can contact Paul directly at PSheriff@pdsa.com.

Top of Page

Print E-Mail Add to Favorites

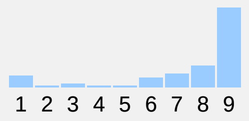
How would you rate the quality of this content?

	1	2	3	4	5	6	7	8	9	
Poor	○	○	○	○	○	○	○	○	○	Outstanding

Tell us why you rated the content this way. (optional)

Submit

Average rating:
7 out of 9



89 people have rated this page