**Syntax for ASP.NET Developers**

[](http://msdn.microsoft.com/en-us/rampup/bb352986.aspx)

Ascend to new heights in your career

[**http://msdn.microsoft.com/rampup**](http://msdn.microsoft.com/rampup)

Contents

[Introduction 3](#_Toc226017431)

[Web Form Syntax 3](#_Toc226017432)

[Server Control Syntax 5](#_Toc226017433)

[Html Server Control Syntax 5](#_Toc226017434)

[Data Binding Syntax <%# %> 6](#_Toc226017435)

[Object Tag Syntax <object runat="server"/> 7](#_Toc226017436)

[Server-Side Comment Syntax <%--Comment --%> 8](#_Toc226017437)

[Server-Side Include Syntax: <-- #Include File="Locaton.inc" --> 8](#_Toc226017438)

[Expression Syntax <%$...%> 9](#_Toc226017439)

# Introduction

ASP.NET syntax defines the structure, layout, and settings of an ASP.NET page. It also defines the layout for ASP.NET server controls, application code, application configuration, and XML Web services. Any ASP.NET feature can be created in text files with the proper file name extension. These include ASP.NET configuration files, application (global.asax) files, pages, and ASP.NET Web server controls.The Visual Studio Express Editions family provides a free, lightweight, and easy-to-learn suite

# Web Form Syntax

An ASP.NET Web Forms page is a declarative text file with an .aspx file name extension. In addition to static content, you can use eight distinct syntax markup elements. This section of the QuickStart reviews each of these syntax elements and provides examples demonstrating their use.

**Rendering Code Syntax: <% %> and <%= %>**

Code rendering blocks are denoted with <% ... %> elements, allow you to custom-control content emission, and execute during the render phase of Web Forms page execution. The following example demonstrates how you can use them to loop over HTML content.  
  
<% For I=0 To 7 %>

<font size="<%=i%>"> Hello World! </font> <br>

<% Next %>  
  
Code enclosed by <% ... %> is just executed, while expressions that include an equal sign, <%= ... %>, are evaluated and the result is emitted as content. Therefore <%="Hello World" %> renders the same thing as the C# code <% Response.Write("Hello World"); %>.   
  
**Note:** For languages that use marks to end or separate statements (for example, the semicolon (;) in C#), it is important to place those marks correctly depending on how your code should be rendered.

|  |  |
| --- | --- |
| **C# code** | |
| <% Response.Write("Hello World"); %> | A semicolon is necessary to end the statement. |
| <%="Hello World"; %> | Wrong: Would result in "Response.Write("Hello World";);". |
| <%="Hello World" %> | A semicolon is not necessary. |

**VB Syntax  
  
<%@ Page Language="VB" %>**

<html>

<body>

<% Dim I As Integer

For I = 0 To 7 %>

<font size="<%=I%>"> Hello World! </font> <br>

<% Next %>

</body>

</html>

**Declaration Code Syntax: <script runat="server">**

Code declaration blocks define member variables and methods that will be compiled into the generated Page class. These blocks can be used to author page and navigation logic. The following example demonstrates how a Subtract method can be declared within a <script runat="server"> block, and then invoked from the page.  
  
**VB**

<script language="VB" runat=server>

Function Subtract(num1 As Integer, num2 As Integer) As Integer

Return(num1 - num2)

End Function

</script>

<%

...

number = subtract(number, 1)

...

%>

**C#**<script language="C#" runat=server>  
int subtract(int num1, int num2) {  
return num1 - num2  
}  
</script>

<%

...

number = subtract(number, 1);

...

%>

**Important:** Unlike ASP -- where functions could be declared within **<% %>** blocks -- all functions and global page variables must be declared in a **<script runat=server>** tag. Functions declared within **<% %>** blocks will now generate a syntax compile error.

## Server Control Syntax

Custom ASP.NET server controls enable page developers to dynamically generate HTML user interface (UI) and respond to client requests. They are represented within a file using a declarative, tag-based syntax. These tags are distinguished from other tags because they contain a **"runat=server"** attribute. The following example demonstrates how an **<asp:label runat="server">** server control can be used within an ASP.NET page. This control corresponds to the **Label** class in the **System.Web.UI.WebControls** namespace, which is included by default.   
  
By adding a tag with the ID "Message", an instance of **Label** is created at run time:

<asp:label id="Message" font-size=24 runat="server"/>

The control can then be accessed using the same name. The following line sets the **Text** property of the control.  
  
**VB**Message.Text = "Welcome to ASP.NET"

**C#**Message.Text = "Welcome to ASP.NET";

## Html Server Control Syntax

HTML server controls enable page developers to programmatically manipulate HTML elements within a page. An HTML server control tag is distinguished from client HTML elements by means of a **"runat=server"** attribute. The following example demonstrates how an HTML **<span runat=server>** server control can be used within an ASP.NET page.   
  
As with other server controls, the methods and properties are accessible programmatically, as shown in the following example.

**VB**<script language="VB" runat="server">

Sub Page\_Load(sender As Object, e As EventArgs)

Message.InnerHtml = "Welcome to ASP.NET"

End Sub

</script>

...

<span id="Message" style="font-size:24" runat="server"/>

**C#**

<script language="C#" runat="server">

void Page\_Load(Object sender, EventArgs e) {

Message.InnerHtml = "Welcome to ASP.NET";

}

</script>

...

<span id="Message" style="font-size:24" runat="server"/>

## Data Binding Syntax <%# %>

The data binding support built into ASP.NET enables page developers to hierarchically bind control properties to data container values. Code located within a <%# %> code block is only executed when the **DataBind** method of its parent control container is invoked. The following example demonstrates how to use the data binding syntax within an **<asp:datalist runat=server>** control.   
  
Within the datalist, the template for one item is specified. The content of the item template is specified using a data binding expression and the Container.DataItem refers to the data source used by the datalist MyList.

<asp:datalist id="MyList" runat=server>

<ItemTemplate>

Here is a value: <%# Container.DataItem %>

</ItemTemplate>

</asp:datalist>

In this case the data source of the MyList control is set programmatically, and then DataBind() is called.

**VB**Sub Page\_Load(sender As Object, e As EventArgs)

Dim items As New ArrayList()

items.Add("One")

items.Add("Two")

items.Add("Three")

MyList.DataSource = items

MyList.DataBind()

End Sub

**C#**

void Page\_Load(Object sender, EventArgs e) {

ArrayList items = new ArrayList();

items.Add("One");

items.Add("Two");

items.Add("Three");

MyList.DataSource = items;

MyList.DataBind();

}

Calling the DataBind method of a control causes a recursive tree walk from that control on down in the tree; the DataBinding event is raised on each server control in that hierarchy, and data binding expressions on the control are evaluated accordingly. So, if the DataBind method of the page is called, then every data binding expression within the page will be called.  
ASP.NET 2.0 also includes a new simplified databinding syntax, that allows controls to automatically data-bind to data source controls, without requiring you to call DataBind() in page code.

## Object Tag Syntax <object runat="server"/>

Object tags enable page developers to declare and create instances of variables using a declarative, tag-based syntax. The following example demonstrates how the object tag can be used to create an instance of an **ArrayList** class.

<object id="items" class="System.Collections.ArrayList" runat="server"/>

The object will be created automatically at run time and can then be accessed through the ID "items".  
**VB**  
Sub Page\_Load(sender As Object, e As EventArgs)

items.Add("One")

items.Add("Two")

items.Add("Three")

...

End Sub

**C#**void Page\_Load(Object sender, EventArgs e) {  
 items.Add("One");  
 items.Add("Two");  
 items.Add("Three");  
 ...   
}

## Server-Side Comment Syntax <%--Comment --%>

Server-side comments enable page developers to prevent server code (including server controls) and static content from executing or rendering. The following sample demonstrates how to block content from executing and being sent down to a client. Note that everything between <%-- and --%> is filtered out and only visible in the original server file, even though it contains other ASP.NET directives.  
  
**VB**<%--

<asp:calendar id="MyCal" runat=server/>

<% For I=0 To 44 %>

Hello World <br>

<% Next %>

--%>

**C#**<%--

<asp:calendar id="MyCal" runat=server/>

<% for (int i=0; i<45; i++) { %>

Hello World <br>

<% } %>

--%>

## Server-Side Include Syntax: <-- #Include File="Locaton.inc" -->

Server-side #Includes enable developers to insert the raw contents of a specified file anywhere within an ASP.NET page. The following sample demonstrates how to insert a custom header and footer within a page.

<!-- #Include File="Header.inc" -->

...

<!-- #Include File="Footer.inc" -->

## Expression Syntax <%$...%>

ASP.NET 2.0 adds a new declarative expression syntax for substituting values into a page before the page is parsed. This is useful for substituting connection string values or application settings defined in a Web.config file for server control property values. It can also be used to substitute values from a resource file for locaization.

<asp:SqlDataSource ID="SqlDataSource1" ConnectionString='<%$ connectionStrings:Pubs %>' runat="server" SelectCommand="sp\_GetAuthors" />

<asp:Label ID="Label1" Text='<%$ Resources: ExchRate, ConvertLabel %>' runat="server"/>