**Introduction to ASP.NET syntax**





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

Contents

[Introduction 3](#_Toc226022409)

[ASP.NET Web Page Syntax Overview 3](#_Toc226022410)

[Example ASP.NET Web Page 4](#_Toc226022411)

[Embedded Code Blocks in ASP.NET Web Pages 9](#_Toc226022412)

# Introduction

ASP.NET is a unified Web development model that includes the services necessary for you to build enterprise-class Web applications with a minimum of coding. ASP.NET is part of the .NET Framework, and when coding ASP.NET applications you have access to classes in the .NET Framework. You can code your applications in any language compatible with the common language runtime (CLR), including Microsoft Visual Basic, C#, JScript .NET, and J#. These languages enable you to develop ASP.NET applications that benefit from the common language runtime, type safety, inheritance, and so on.Visual Studio Web Development tools

## ASP.NET Web Page Syntax Overview

ASP.NET Web pages are created in a manner similar to static HTML Web pages (pages that do not include server-based processing), but they include extra elements that ASP.NET recognizes and processes when the page runs. The characteristics that distinguish ASP.NET Web pages from static HTML (or other) pages are as follows:

* A file name extension of .aspx instead of .htm, .html, or other file name extensions. The .aspx file name extension causes the page to be processed by ASP.NET. The mapping of file name extensions to ASP.NET is done in Internet Information Services (IIS). By default, .aspx pages are run by ASP.NET and .htm and .html pages are not.
* An optional @ Page directive or other directive, as appropriate to the type of page that you are creating.
* A **form** element that is configured correctly for ASP.NET. The **form** element is required only if the page contains controls whose values you want to use during page processing.
* Web server controls.
* Server code, if you add your own code to the page.
* If you want your pages to conform to XHTML standards, you must include additional elements, such as a DOCTYPE element.

### Example ASP.NET Web Page

The following code example shows a page that includes the basic elements that constitute an ASP.NET Web page. The page contains static text as you might have in an HTML page, along with elements that are specific to ASP.NET. The elements that are specific to ASP.NET are highlighted.

**VB**

<b><%@ Page Language="VB" %></b>

<html>

<b><script runat="server"></b>

<b> Sub Button1\_Click(ByVal sender As Object, \_</b>

<b> ByVal e As System.EventArgs)</b>

<b> Label1.Text = "Welcome, " & TextBox1.Text</b>

<b> End Sub</b>

<b></script></b>

<head <b>runat="server"</b>>

 <title>Basic ASP.NET Web Page</title>

</head>

<body>

 <form id="form1" <b>runat="server"</b>>

 <h1>Welcome to ASP.NET</h1>

 <p>Type your name and click the button.</p>

 <p>

 <b><asp:TextBox ID="TextBox1" runat="server"></asp:TextBox></b>

 <b><asp:Button ID="Button1" runat="server" </b>

 <b>Text="Click" OnClick="Button1\_Click" /></b>

 </p>

 <p>

 <b><asp:Label ID="Label1" runat="server"></asp:Label></b>

 </p>

 </form>

</body>

</html>

C#
<b><%@ Page Language="C#" %></b>

<html>

<b><script runat="server"></b>

<b> void Button1\_Click(Object sender, EventArgs e)</b>

<b> Label1.Text = "Welcome, " + TextBox1.Text;</b>

<b> End Sub</b>

<b></script></b>

<head runat="server">

 <title>Basic ASP.NET Web Page</title>

</head>

<body>

 <form id="form1" <b>runat="server"</b>>

 <h1>Welcome to ASP.NET</h1>

 <p>Type your name and click the button.</p>

 <p>

<b> <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox></b>

<b> <asp:Button ID="Button1" runat="server" </b>

<b> Text="Click" OnClick="Button1\_Click" /></b>

 </p>

 <p>

<b> <asp:Label ID="Label1" runat="server"></asp:Label></b>

 </p>

 </form>

</body>

</html>

#### @ Directives

ASP.NET pages usually contain directives that allow you to specify page properties and configuration information for the page. The directives are used by ASP.NET as instructions for how to process the page, but they are not rendered as part of the markup that is sent to the browser.

The most commonly used directive is the @ Page directive, which allows you to specify many configuration options for the page, including the following:

* The server programming language for code in the page.
* Whether the page is a page with server code directly in the page, which is called a single-file page, or whether it is a page with code in a separate class file, which is called a code-behind page. In the previous example, the page is a single-file page; the code is directly in the page, and the **@ Page** directive does not include information about linked class files. For more information, see the "Server Code" section later in this topic.
* Debugging and tracing options.
* Whether the page has an associated master page and should therefore be treated as a content page.

If you do not include an **@ Page** directive in the page, or if the directive does not include a specific setting, settings are inherited the from the configuration file for the Web application (the Web.config file) or from the site configuration file (the Machine.config file).

In addition to including an **@ Page** directive, you can include other directives that support additional page-specific options. Other common directives include the following:

* @ Import   This directive allows you to specify namespaces that you want to reference in your code.
* @ OutputCache   This directive allows you to specify that the page should be cached, along with parameters for when and how long to cache the page.
* @ Implements   This directive allows you to specify that the page implement a .NET interface.
* @ Register   This directive allows you to register additional controls for use on the page. The **@ Register** directive declares the control's tag prefix and the location of the control's assembly. You must use this directive if you want to add user controls or custom ASP.NET controls to a page.

Certain types of ASP.NET files use a directive other than **@ Page**. For example, ASP.NET master pages use an @ Master directive, and ASP.NET user controls use an @ Control directive. Each directive allows you to specify different options that are appropriate for the file.

#### Form Elements

If your page includes controls that allow users to interact with the page and submit it, the page must include a **form** element. You use the standard HTML **form** element, but certain rules apply. The rules for using the **form** element are as follows:

* The page can contain only one **form** element.
* The **form** element must contain the **runat** attribute with the value set to **server**. This attribute allows you to refer to the form and the controls on the page programmatically in server code.
* Server controls that can perform a postback must be inside the **form** element.
* The opening tag must not contain an **action** attribute. ASP.NET sets these attributes dynamically when the page is processed, overriding any settings that you might make.

#### Web Server Controls

In most ASP.NET pages, you will add controls that allow the user to interact with the page, including buttons, text boxes, lists, and so on. These Web server controls are similar to HTML buttons and **input** elements. However, they are processed on the server, allowing you to use server code to set their properties. These controls also raise events that you can handle in server code.

Server controls use a special syntax that ASP.NET recognizes when the page runs. The following code example shows some typical Web server controls.

<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>

<asp:Button ID="Button1" runat="server" Text="Click" OnClick="Button1\_Click" />

The tag name for ASP.NET server controls starts with a prefix — in this case, asp:. The prefix might be different if the control is not part of the .NET Framework. ASP.NET server controls also include the **runat="server"** attribute and, optionally, an ID that you can use to reference the control in server code.

When the page runs, it identifies the server controls and runs the code that is associated with those controls. Many controls render some HTML or other markup into the page. For example, the **asp:textbox** control renders an **input** element with the **type="text"** attribute into a page. However, there is not necessarily a one-to-one mapping between a Web server control and an HTML element. For example, the **asp:calendar** control renders an HTML table. Some controls do not render anything to the browser; instead, they are processed on the server only, and they provide information to other controls.

#### HTML Elements as Server Controls

Instead of, or in addition to, using ASP.NET server controls, you can use ordinary HTML elements as server controls. You can add the **runat="server"** attribute and an **ID** attribute to any HTML element in the page. When the page runs, ASP.NET identifies the element as a server control and makes it available to server code. For example, you can add the required elements to an HTML **body** element, as shown in the following code example.

<body runat="server" id="body">

You can then reference the **body** element in server code — for example, to set the body background color at run time in response to user input or to information from a database.

#### Server Code

Most ASP.NET pages include code that runs on the server when the page is processed. ASP.NET supports many languages including C#, Visual Basic, J#, Jscript, and others.

ASP.NET supports two models for writing server code for a Web page. In the single-file model, the code for the page is in a **script** element where the opening tag includes the **runat="server"** attribute. The example earlier in this topic shows the single-file model.

Alternatively, you can create the code for the page in a separate class file, which is referred to as the code-behind model. In this case, the ASP.NET Web page generally contains no server code. Instead, the @ Page directive includes information that links the .aspx page with its associated code-behind file. The following code example shows a typical **@ Page** directive for a page with a code-behind file.

**VB**
<%@ Page Language="VB" CodeFile="Default.aspx.vb" Inherits="Default" %>

**C#**

<%@ Page Language="C#" CodeFile="Default.aspx.cs" Inherits="Default" %>

The **CodeFile** attribute specifies the name of the separate class file, and the **Inherits** attribute specifies the name of the class within the code-behind file that corresponds to the page.

## Embedded Code Blocks in ASP.NET Web Pages

The default model for adding code to an ASP.NET Web page is to either create a code-behind class file (a code-behind page) or to write the page's code in a **script** block with the attribute **runat="server"** (a single-file page). The code you write typically interacts with controls on the page. For example, you can display information on the page from code by setting the **Text** (or other) properties of controls.

Another possibility is to embed code directly into the page using an embedded code block.

#### Embedded Code Blocks

An embedded code block is server code that executes during the page's render phase. The code in the block can execute programming statements and call functions in the current page class.

The following code example shows an ASP.NET page with an embedded code block that displays the results of a loop.
**VB**<%@ Page Language="VB" %>

<html>

<body>

 <form id="form1" runat="server">

<b> <% For i As Integer = 0 To 5 %></b>

<b> <% Response.Write("<br>" & i.ToString())%></b>

<b> <% Next%></b>

 </form>

</body>

</html>

**C#**

<%@ Page Language="C#" %>

<html>

<body>

 <form id="form1" runat="server">

<b> <% for(int i = 0; i < 6; i++) %></b>

<b> <% { Response.Write("<br>" + i.ToString()); }%></b>

 </form>

</body>

</html>

The following code example shows an embedded code block that displays the value of a public GetTime() function inside a span element. In embedded code blocks, the syntax <% = expression %> is used to resolve an expression and return its value into the block.
**VB**<%@ Page Language="VB" %>

<script runat=server>

Protected Function GetTime() As String

 Return DateTime.Now.ToString("t")

End Function

</script>

<html>

<body>

 <form id="form1" runat="server">

 Current server time is <b><% =GetTime()%></b>.

 </form>

</body>

</html>

**C#**

<%@ Page Language="C#" %>

<script runat=server>

protected String GetTime()

{

 return DateTime.Now.ToString("t");

}

</script>

<html>

<body>

 <form id="form1" runat="server">

 Current server time is <b><% =GetTime()%></b>.

 </form>

</body>

</html>

Embedded code blocks must be written in the page's default language. For example, if the page's [@ Page](http://msdn.microsoft.com/en-us/library/ydy4x04a%28VS.80%29.aspx) directive contains the attribute language="VB", the page will use the Visual Basic compiler to compile code in any script block marked with runat="server" and any in-line code in <% %> delimiters

#### Uses for Embedded Code Blocks

Embedded code blocks are supported in ASP.NET Web pages primarily to preserve backward compatibility with older ASP technology. In general, using embedded code blocks for complex programming logic is not a best practice, because when the code is mixed on the page with markup, it can be difficult to debug and maintain. In addition, because the code is executed only during the page's render phase, you have substantially less flexibility than with code-behind or script-block code in scoping your code to the appropriate stage of page processing.

Some uses for embedded code blocks include:

* Setting the value of a control or markup element to a value returned by a function, as illustrated in the preceding example.
* Embedding a calculation directly into the markup or control property.