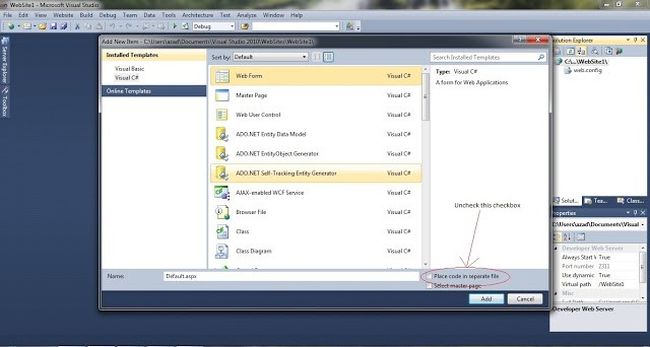
**ASP.NET PAGE STRUCTURE OPTIONS:**

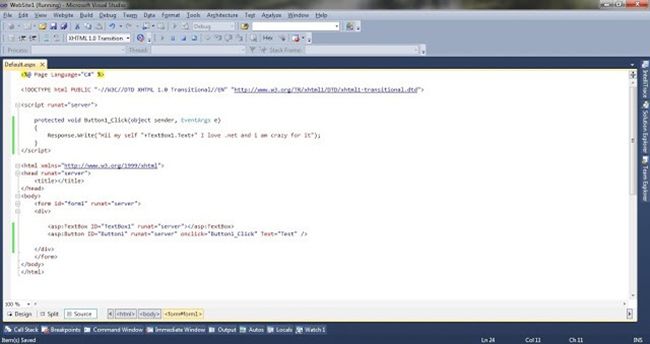
Today I describe the ASP.Net page structure. Which page structure you need to use and what are the differences between them. There are two ways structuring your code in an ASP.Net page. These two ways are:

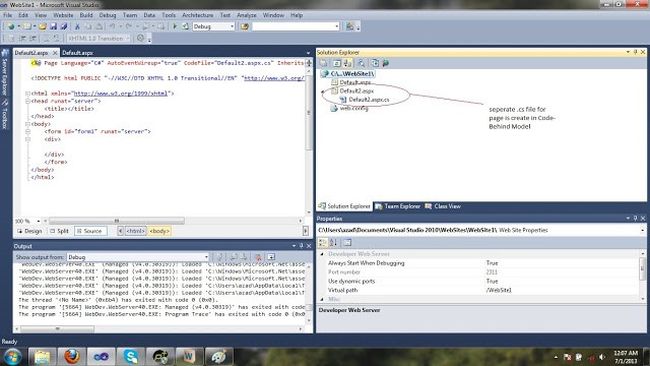
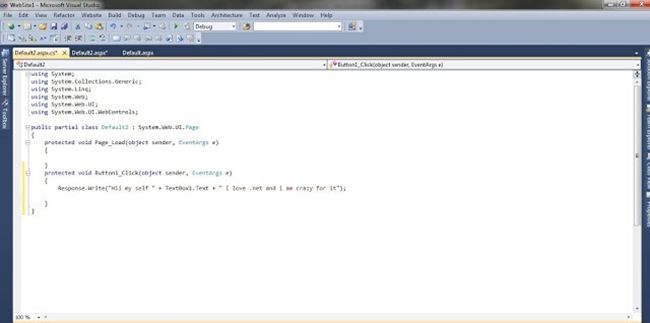
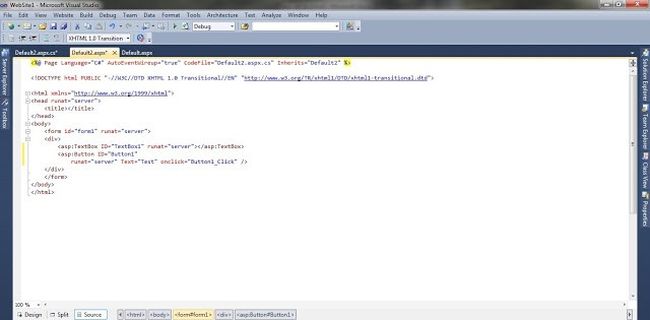
1. Code-Inline Model
2. Code-Behind Model

**First Option is to use Code-Inline Model**  
  
Let us start with the Code-Inline mode. Most ASP.Net 2.0/3.0 developers are familiar with this model. In this model you need to write your code in a single .aspx page. When the developers were working with .Net Framework 1.0/1.1 they move out of their way and start building their ASP.Net pages in Inline-Code model. In Visual Studio you can simply select your page type to build ASP.Net page in Inline Code. For this you need to uncheck the "Place code in separate file" checkbox.  
  
[](https://www.c-sharpcorner.com/UploadFile/azadc/page-structure-in-Asp-Net/Images/ASP.NETStruc1large.jpg)

Here is the list of files for using inline code:

* Web form
* Ajax web form
* Master pages
* Ajax master pages
* Web user controls
* Web services

So now for a little practical information.  
  
Add a TextBox and a button to your page. Now double-click on the button. Now write the following code in the button click:  
  
protected void Button1\_Click(object sender, EventArgs e)  
{  
 Response.Write("Hii my self "+TextBox1.Text+" I love .net and i am crazy for it");  
}  
  
[](https://www.c-sharpcorner.com/UploadFile/azadc/page-structure-in-Asp-Net/Images/ASP.NETStruc1.5large.jpg)

Run your page and check the results.  
  
**The Second Option you can use is a Code-Behind Model**  
  
Now suppose you are working with an inline-model and your code has more than 400-500 lines. In the same page you have the UI of the page. Then how complex is your page now? Do you think about it? So to resolve the complexity issue of a aspx page you can use the Code-Behind Model. For this you need to check the checkbox "Place code in separate file". See in the Solution Explorer, you will see a separate file has been created with a .cs extension.  
  
[](https://www.c-sharpcorner.com/UploadFile/azadc/page-structure-in-Asp-Net/Images/ASP.NETStruc2large.jpg)  
Now you can do the same on this page and notice that the code that you write after a double-click on a button is in another file.  
  
[](https://www.c-sharpcorner.com/UploadFile/azadc/page-structure-in-Asp-Net/Images/ASP.NETStruc3large.jpg)  
Code-Behind just separates your business logic and presentation logic. It is my recommendation to use the Code-Behind Model.  
  
I just want to add something extra. Notice in your aspx file, there are some attributes in the page directive, as in the following:  
  
[](https://www.c-sharpcorner.com/UploadFile/azadc/page-structure-in-Asp-Net/Images/ASP.NETStruc4large.jpg)  
First is "Codefile", this attribute is for your .cs file. It points to the cs file of your aspx page. The second attribute is "Inherit". It specifies the name of the class that is bound to the page when the page is compiled.

**ASP.net 4 page Directives:**

ASP.NET directives are instructions to specify optional settings, such as registering a custom control and page language. These settings describe how the web forms (.aspx) or user controls (.ascx) pages are processed by the .Net framework.

The syntax for declaring a directive is:

<%@ directive\_name attribute=value [attribute=value] %>

In this section, we will just introduce the ASP.NET directives and we will use most of these directives throughout the tutorials.

The Application Directive

The Application directive defines application-specific attributes. It is provided at the top of the global.aspx file.

The basic syntax of Application directive is:

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

The attributes of the Application directive are:

|  |  |
| --- | --- |
| **Attributes** | **Description** |
| Inherits | The name of the class from which to inherit. |
| Description | The text description of the application. Parsers and compilers ignore this. |
| Language | The language used in code blocks. |

The Assembly Directive

The Assembly directive links an assembly to the page or the application at parse time. This could appear either in the global.asax file for application-wide linking, in the page file, a user control file for linking to a page or user control.

The basic syntax of Assembly directive is:

<%@ Assembly Name ="myassembly" %>

The attributes of the Assembly directive are:

|  |  |
| --- | --- |
| **Attributes** | **Description** |
| Name | The name of the assembly to be linked. |
| Src | The path to the source file to be linked and compiled dynamically. |

The Control Directive

The control directive is used with the user controls and appears in the user control (.ascx) files.

The basic syntax of Control directive is:

<%@ Control Language="C#" EnableViewState="false" %>

The attributes of the Control directive are:

|  |  |
| --- | --- |
| **Attributes** | **Description** |
| AutoEventWireup | The Boolean value that enables or disables automatic association of events to handlers. |
| ClassName | The file name for the control. |
| Debug | The Boolean value that enables or disables compiling with debug symbols. |
| Description | The text description of the control page, ignored by compiler. |
| EnableViewState | The Boolean value that indicates whether view state is maintained across page requests. |
| Explicit | For VB language, tells the compiler to use option explicit mode. |
| Inherits | The class from which the control page inherits. |
| Language | The language for code and script. |
| Src | The filename for the code-behind class. |
| Strict | For VB language, tells the compiler to use the option strict mode. |

The Implements Directive

The Implement directive indicates that the web page, master page or user control page must implement the specified .Net framework interface.

The basic syntax for implements directive is:

<%@ Implements Interface="interface\_name" %>

The Import Directive

The Import directive imports a namespace into a web page, user control page of application. If the Import directive is specified in the global.asax file, then it is applied to the entire application. If it is in a page of user control page, then it is applied to that page or control.

The basic syntax for import directive is:

<%@ namespace="System.Drawing" %>

The Master Directive

The Master directive specifies a page file as being the mater page.

The basic syntax of sample MasterPage directive is:

<%@ MasterPage Language="C#" AutoEventWireup="true" CodeFile="SiteMater.master.cs" Inherits="SiteMaster" %>

The MasterType Directive

The MasterType directive assigns a class name to the Master property of a page, to make it strongly typed.

The basic syntax of MasterType directive is:

<%@ MasterType attribute="value"[attribute="value" ...] %>

The OutputCache Directive

The OutputCache directive controls the output caching policies of a web page or a user control.

The basic syntax of OutputCache directive is:

<%@ OutputCache Duration="15" VaryByParam="None" %>

The Page Directive

The Page directive defines the attributes specific to the page file for the page parser and the compiler.

The basic syntax of Page directive is:

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="\_Default" Trace="true" %>

The attributes of the Page directive are:

|  |  |
| --- | --- |
| **Attributes** | **Description** |
| AutoEventWireup | The Boolean value that enables or disables page events that are being automatically bound to methods; for example, Page\_Load. |
| Buffer | The Boolean value that enables or disables HTTP response buffering. |
| ClassName | The class name for the page. |
| ClientTarget | The browser for which the server controls should render content. |
| CodeFile | The name of the code behind file. |
| Debug | The Boolean value that enables or disables compilation with debug symbols. |
| Description | The text description of the page, ignored by the parser. |
| EnableSessionState | It enables, disables, or makes session state read-only. |
| EnableViewState | The Boolean value that enables or disables view state across page requests. |
| ErrorPage | URL for redirection if an unhandled page exception occurs. |
| Inherits | The name of the code behind or other class. |
| Language | The programming language for code. |
| Src | The file name of the code behind class. |
| Trace | It enables or disables tracing. |
| TraceMode | It indicates how trace messages are displayed, and sorted by time or category. |
| Transaction | It indicates if transactions are supported. |
| ValidateRequest | The Boolean value that indicates whether all input data is validated against a hardcoded list of values. |

The PreviousPageType Directive

The PreviousPageType directive assigns a class to a page, so that the page is strongly typed.

The basic syntax for a sample PreviousPagetype directive is:

<%@ PreviousPageType attribute="value"[attribute="value" ...] %>

The Reference Directive

The Reference directive indicates that another page or user control should be compiled and linked to the current page.

The basic syntax of Reference directive is:

<%@ Reference Page ="somepage.aspx" %>

The Register Directive

The Register derivative is used for registering the custom server controls and user controls.

The basic syntax of Register directive is:

<%@ Register Src="~/footer.ascx" TagName="footer" TagPrefix="Tfooter" %>

### Global.asax:

* The Global.asax is also known as the ASP.NET application file and is used to serve application-level and session-level events.
* It allows us to write code that response to global application events raised by ASP.NET or by HttpModules.
* These events fire at various points during the lifetime of a web application, including when the application domain is first created.
* The Global.asax file resides in the root directory of an ASP.NET-based application
* At run time, global.asax is parsed and compiled into a dynamically generated .NET Framework class derived from the HttpApplication base classThe Global.asax file is optional. If you do not define the file, the ASP.NET page framework assumes that you have not defined any application or session event handlers

### **Basic Application Events:**

|  |  |  |
| --- | --- | --- |
| S. No. | **Event Handling Method** | **Description** |
| 1 | Application\_Start() | Application\_Start() event occurs when the application starts, which is the first time it receives a request from any user. It doesn’t occur on subsequent requests. This event is commonly used to create or cache some initial information that will be reused later. |
| 2 | Application\_End() | Application\_End() event occurs when the application is shutting down, generally because the web server has restarted. You can insert cleanup code here. |
| 3 | Application\_BeginRequest() | Application\_BeginRequest() event occurs with each request the application receives, just before the page code is executed. |
| 4 | Application\_EndRequest() | Application\_EndRequest() event occurs with each request the application receives, just after the page code is executed. |
| 5 | Session\_Start() | Session\_Start() event occurs whenever a new user request is received and a session is started. |
| 6 | Session\_End() | Session\_End() event occurs when a session times out or is programmatically ended. This event is only raised if you are using in-process session state storage (the InProc mode, not the StateServer or SQLServer modes ). |
| 7 | Application\_Error() | Application\_Error() event occurs in response to an un-handled error. |

|  |  |
| --- | --- |
|  |  |