Creating Consistent Looking Web Sites

Asst. Prof. Dr. Özgü Can

Consistent Pages

- Make your web site as consistent as possible.

- Consistency gives your site a **professional appearance** and it **helps your visitors to find their way** around the site.

Master Pages

- With most web sites, **only** <u>part of the page changes</u> when you go from one page to another.
- The parts that don't change usually include common regions like the *header*, a *menu*, and the *footer*.
- To create web pages with a consistent layout you need a way to define these relatively static regions in a **single template file**.

Master Pages

- You **only** <u>need to modify the master page</u> and the pages based on this master will pick up the changes **automatically**.
- Looks like a normal ASPX page
  - Not a true ASPX page
    - **Can not be** requested in the browser.
    - It **only** serves as the **template** that real web pages (*content pages)* are based on.

Master Pages

- Instead of the **@ Page** directive, a master page uses an **@ Master** directive that identifies the file as a master page:

```
<%@ Master Language="C#" %>
```

Master Pages

- Just like a normal ASPX page, a master page can have a **Code Behind** file, identified by its **`CodeFile`** and **`Inherits`** attributes:

```
<%@ Master Language="C#" AutoEventWireup="true"
    CodeFile="Frontend.master.cs" Inherits="MasterPages_Frontend" %>
```

Master Pages

- To create regions that content pages can fill in, you need to define **`ContentPlaceHolder`** controls in your page like this:

```
<asp:ContentPlaceHolder id="ContentPlaceholder1" runat="server">

</asp:ContentPlaceHolder>
```

- You can create as many placeholders as you like.

Content Pages

- Normal ASPX files
- Connected to a master page using the **`MasterPageFile`** attribute of the **`Page`** directive:

```
<%@ Page Title=" " Language="C#" MasterPageFile="~/MasterPages/Frontend.master"
    AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>
```

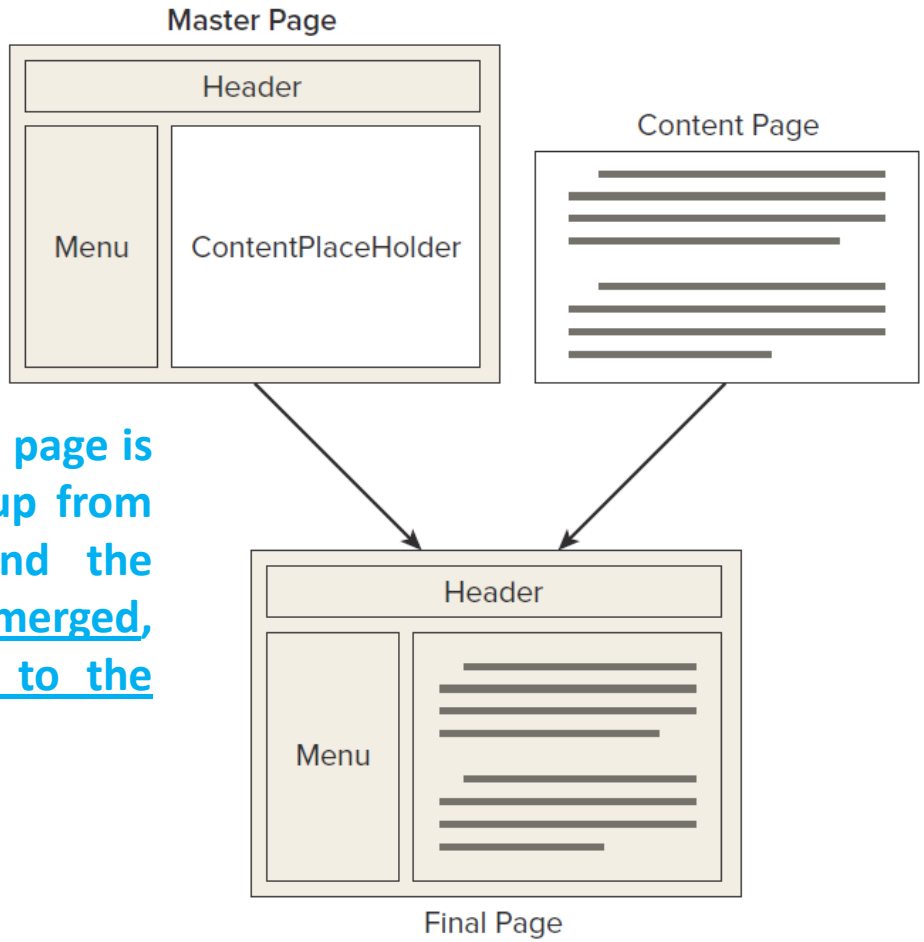
Content Pages

- The **page-specific content** is then <u>put inside</u> a **`Content`** control that points to the relevant **`ContentPlaceHolder`**:

```
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
</asp:Content>
```

Points to the `ContentPlaceHolder` that is defined in the master page.

Runtime



At runtime, when the page is requested, the markup from the master page and the content page are merged, processed, and sent to the browser.

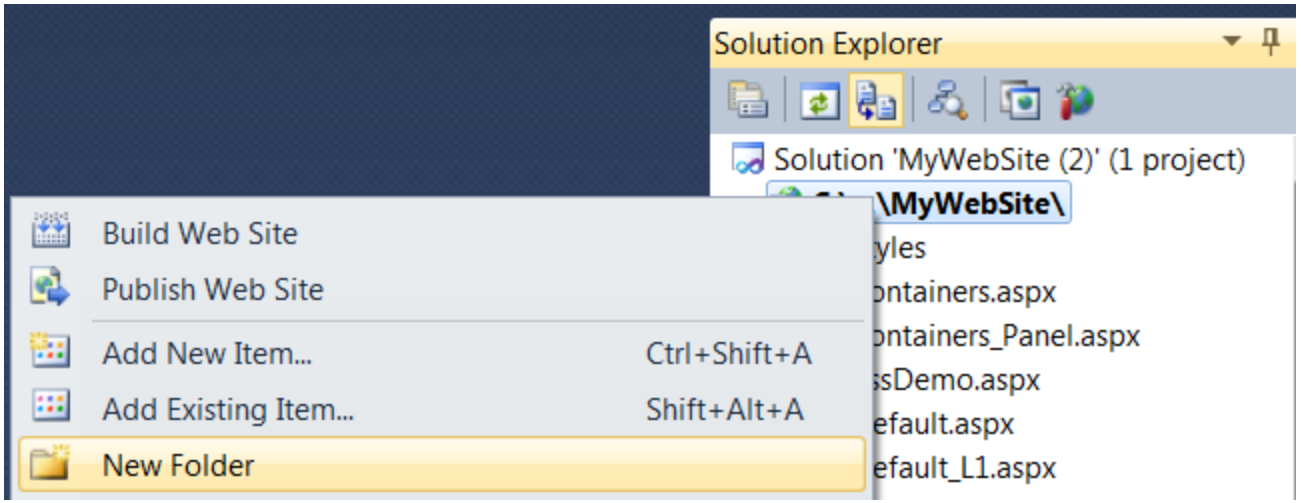
Master Pages

- It’s often easier to store them in a separate folder.

- Just like normal ASPX pages, they support the ***Inline Code Model*** as well as the ***Code Behind Model***.

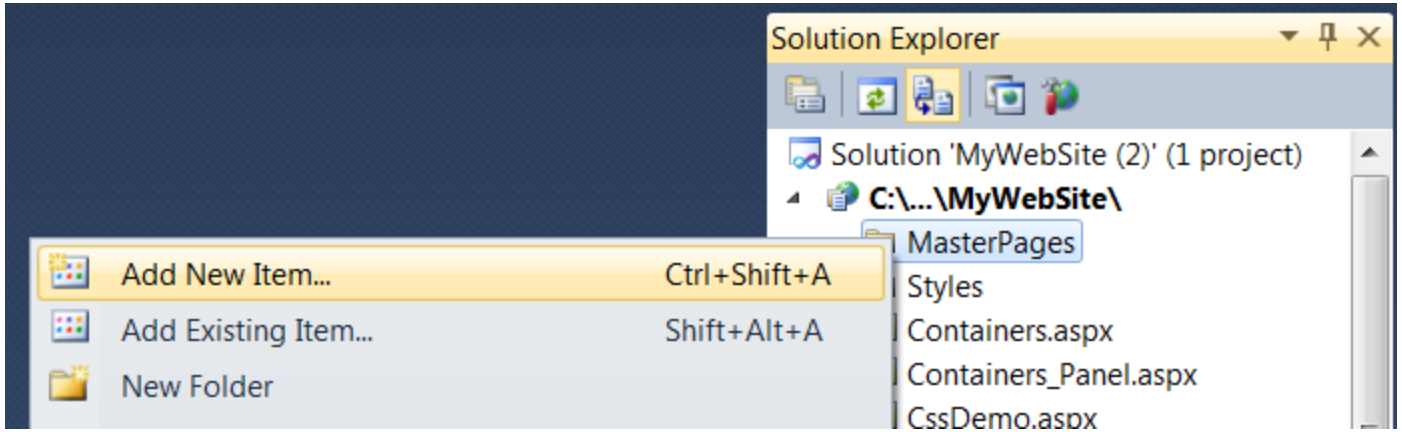
Example
Master Page

- Add **`MasterPages`** folder

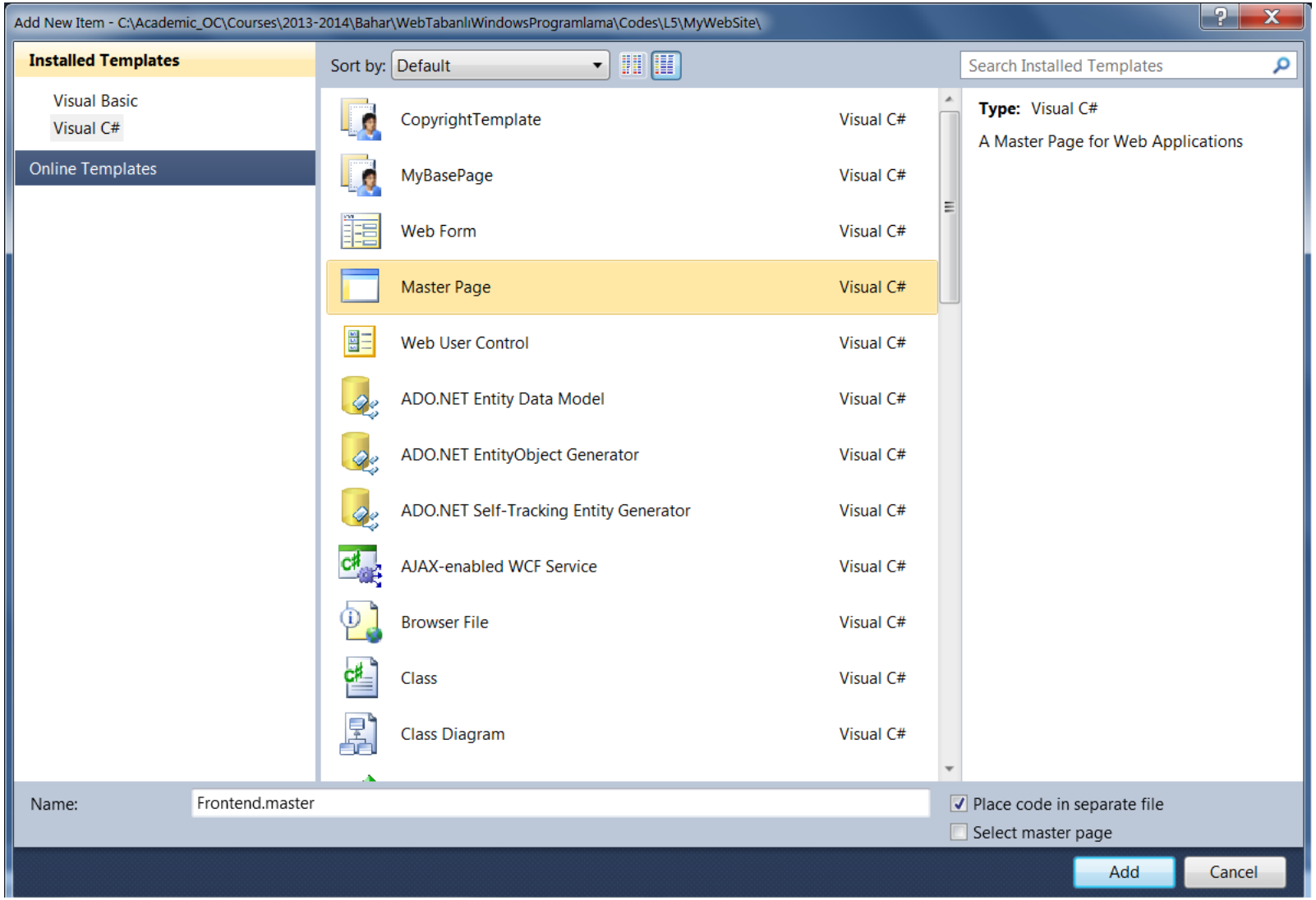


Example
Master Page

- Add New Item to the **`MasterPages`** folder



Example
Master Page

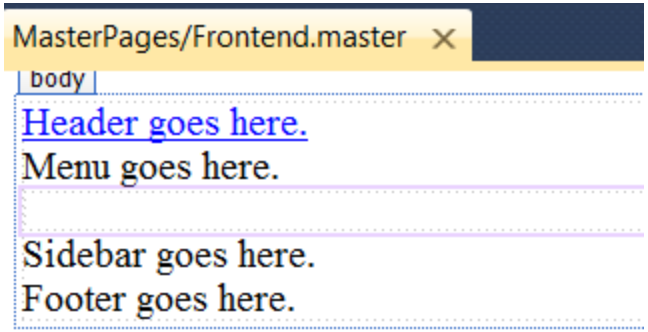


Example
Master Page

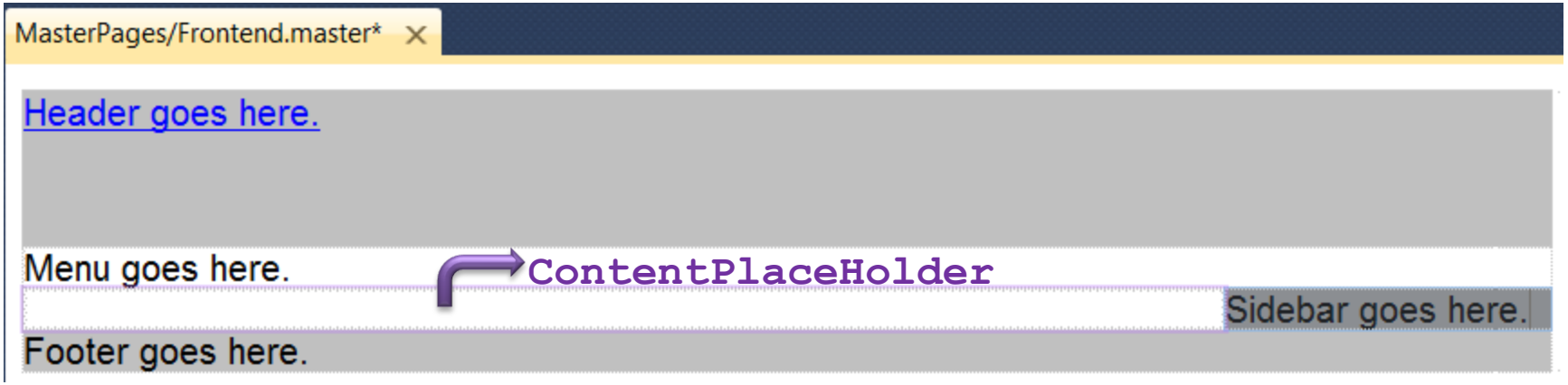
```
<form id="form1" runat="server">
<div id="PageWrapper">
        <div id="Header"><a href="~/" runat="server">Header goes here.</a></div>
        <div id="MenuWrapper">Menu goes here.</div>
        <div id="MainContent">
            <asp:ContentPlaceHolder id="cpMainContent" runat="server">

            </asp:ContentPlaceHolder>
        </div>
        <div id="Sidebar">Sidebar goes here.</div>
        <div id="Footer">Footer goes here.</div>
    </div>
</form>
```

Example
Master Page

- In Design View:



- Drag **`Styles.css`** from the **`Styles`** folder in the Solution Explorer onto the master page.



Example
Adding a Content Page

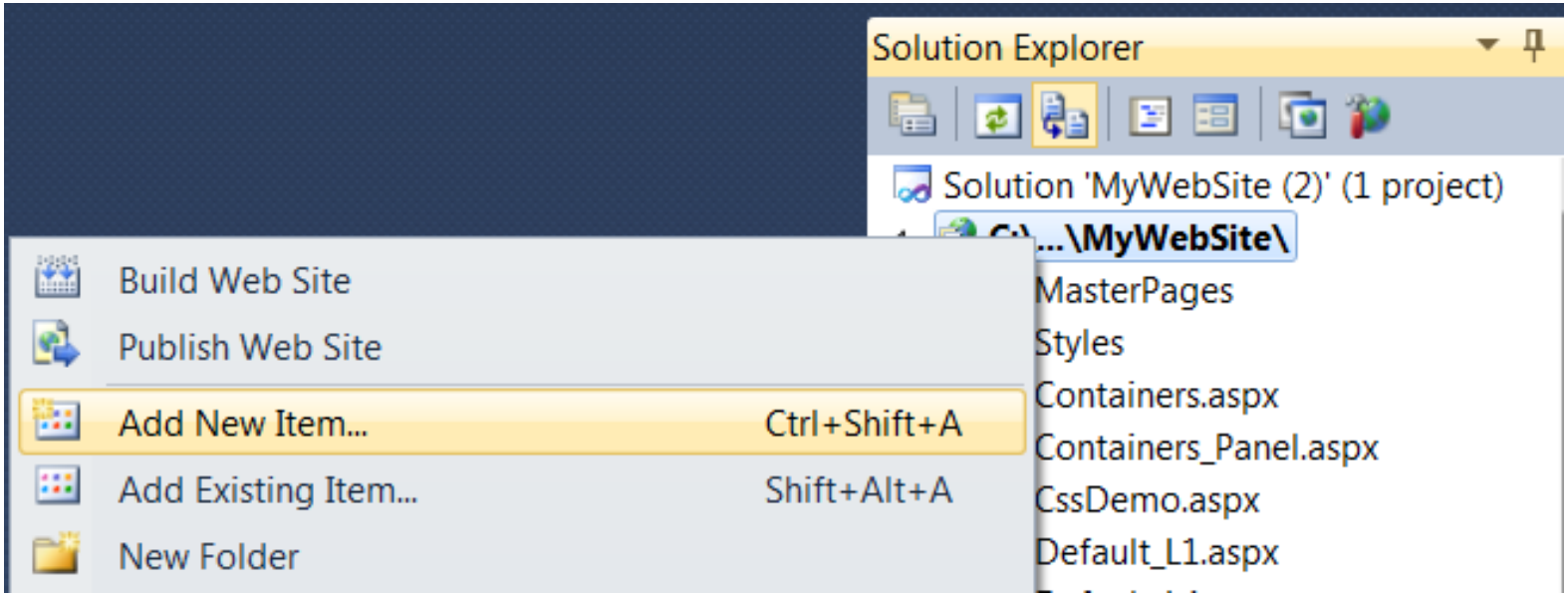
- In **`Default.aspx`** → Copy all the HTML between **`MainContent <div>`** tags

```
<div id="MainContent">
    <h1 style="padding: 0px; margin: 0px 0px 10px 0px">
        Hi there visitor! Welcome to my Web Site :-)</h1>
    <p class="Introduction">
        I&#39;m glad that you are visiting my web site <a href="http://www.mywebsite.com">
        http://www.mywebsite.com</a>
    </p>
    <p>
        <strong>Feel</strong> <span class="style1"><strong>free</strong></span> to have a
        <a href="Default.aspx">look around</a>.</p>
</div>
```

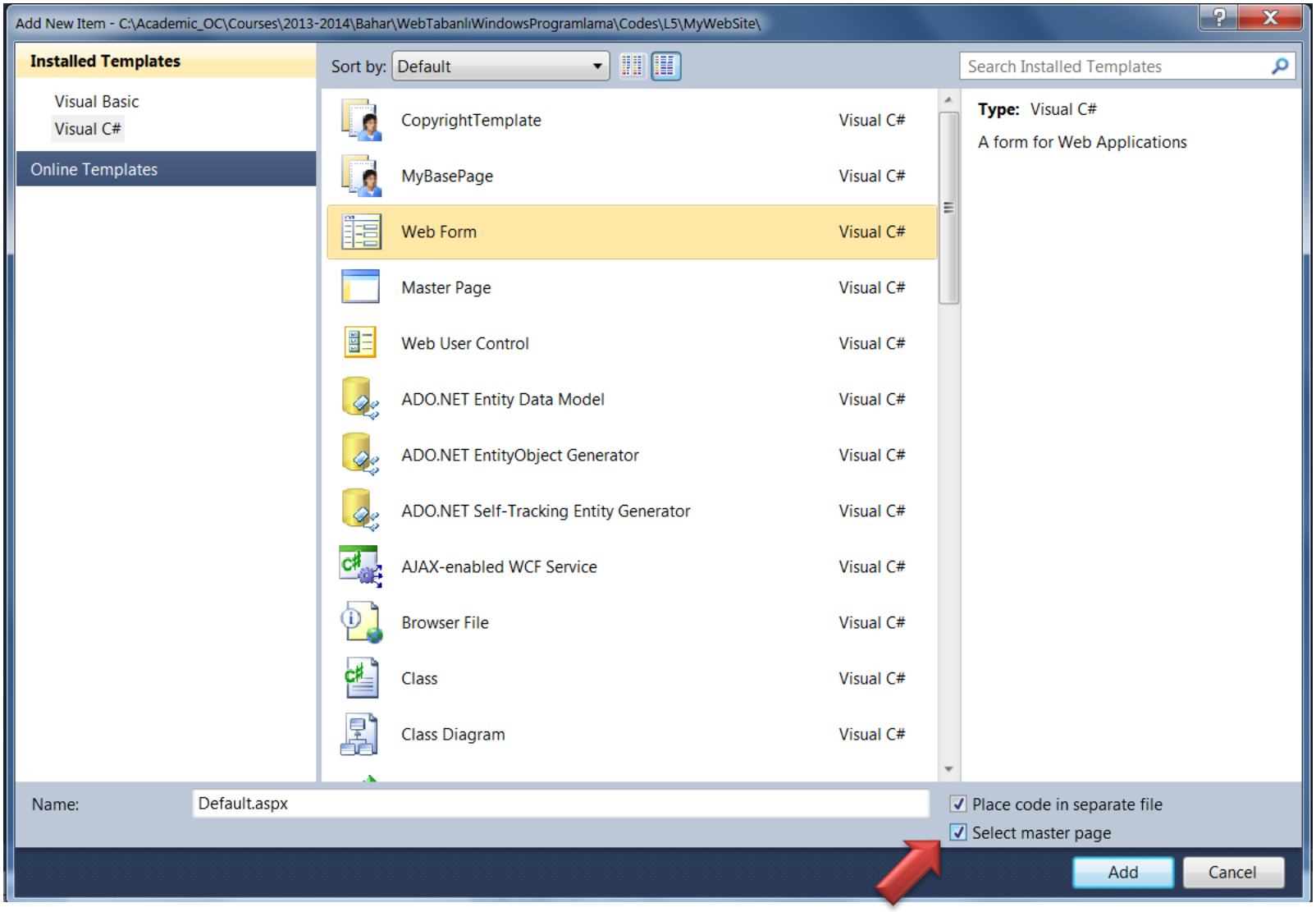
- Delete/Rename **`Default.aspx`**

Example
Adding a Content Page

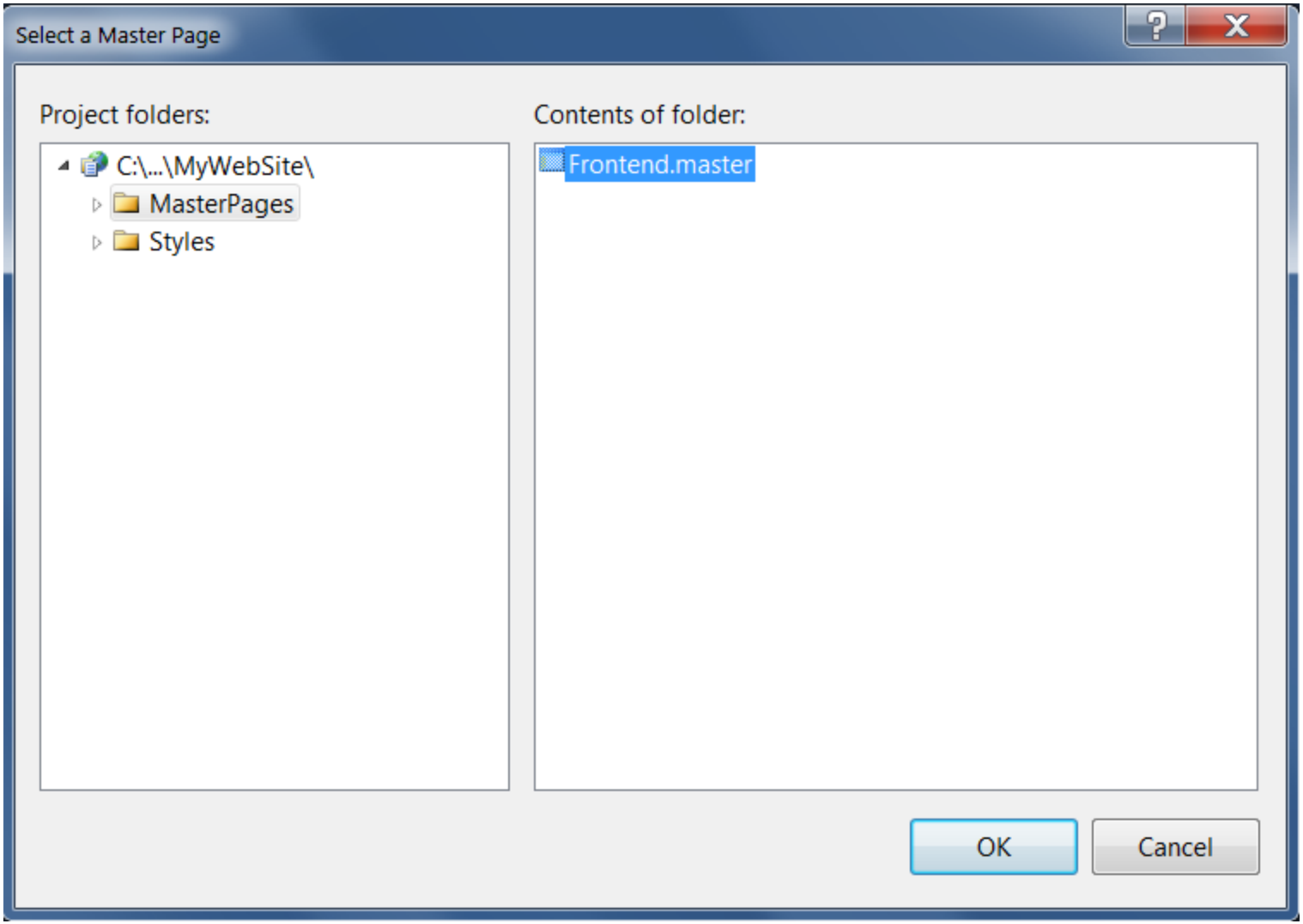
- Add New Item



Example
Adding a Content Page



Example
Adding a Content Page



Example
Adding a Content Page

- Instead of getting a full page with HTML as you got with standard ASPX pages, you now only get two **`<asp:Content>`** placeholders

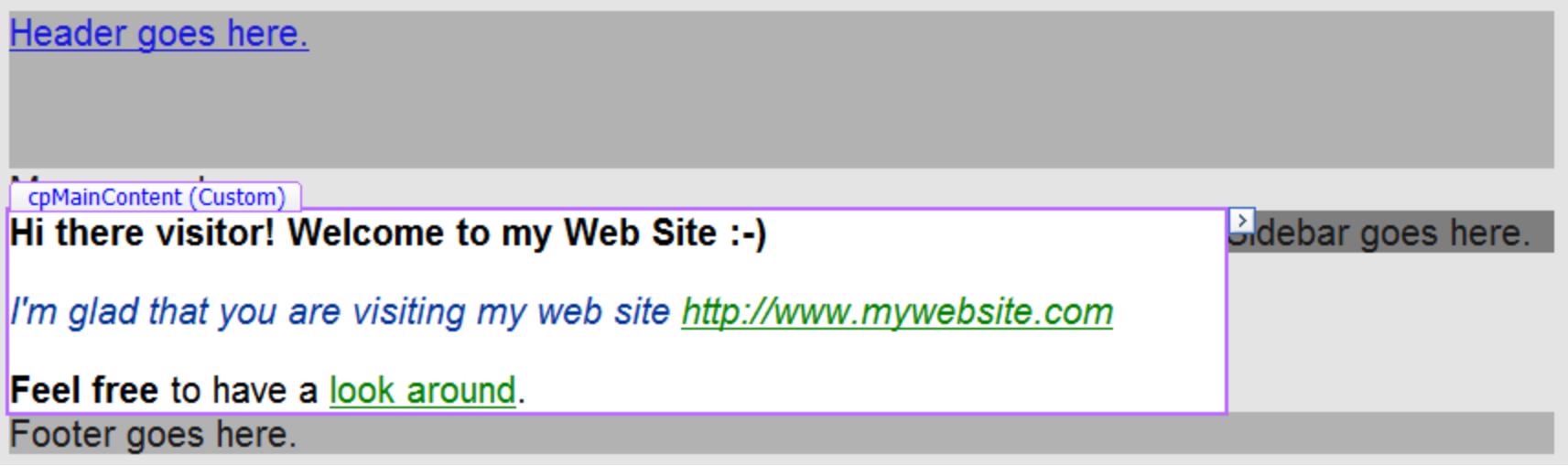
```
<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
</asp:Content>
```

Example
Adding a Content Page

- In Design View → Everything is grayed out and read-only, except for the **`<asp:Content>`** region for **`cpMainContent`**.

Example
Adding a Content Page

- Click once inside the **`cpMainContent`** placeholder and paste the old markup from the **`Default.aspx`**.



Example
Adding a Content Page

- View Source

```
<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
    <h1 style="padding: 0px; margin: 0px 0px 10px 0px">
        Hi there visitor! Welcome to my Web Site :-)</h1>
    <p class="Introduction">
        I&#39;m glad that you are visiting my web site <a href="http://www.mywebsite.com">
        http://www.mywebsite.com</a>
    </p>
    <p>
        <strong>Feel</strong> <span class="style1"><strong>free</strong></span> to have
        a <a href="Default.aspx">look around</a>.</p>
</asp:Content>
```

Example
Adding a Content Page

- View in browser.

Header goes here.

Menu goes here.

Hi there visitor! Welcome to my Web Site :-)

Sidebar goes here.

I'm glad that you are visiting my web site http://www.mywebsite.com

Feel free to have a look around.

Footer goes here.

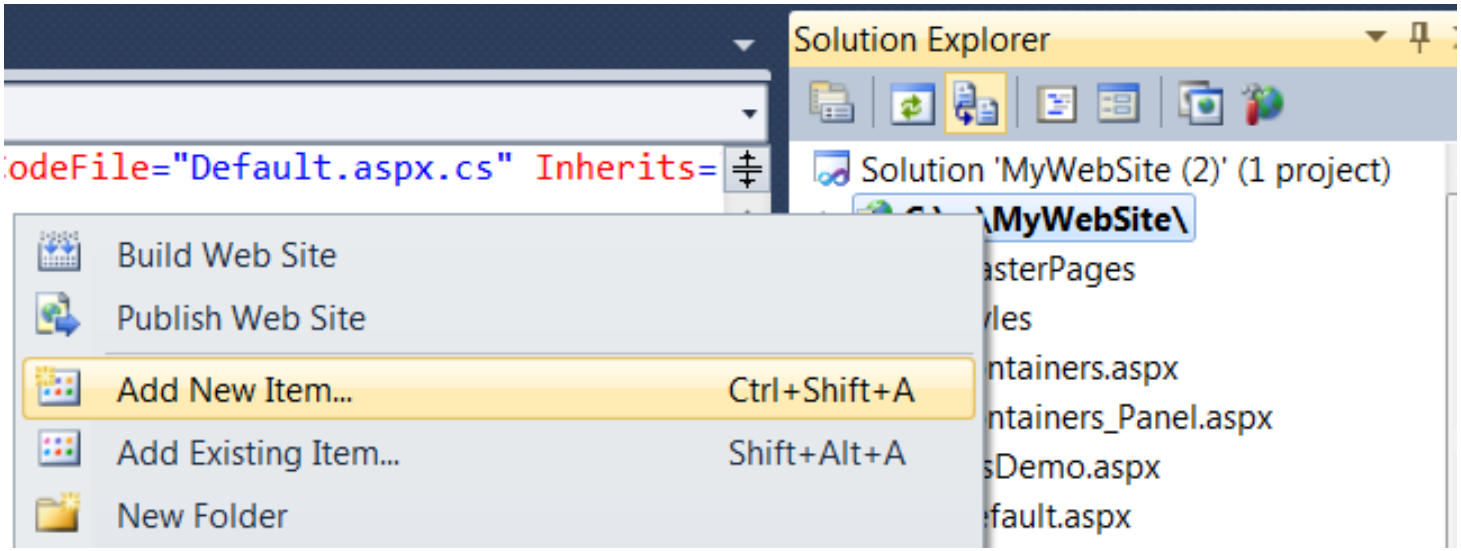
Example
Adding a Content Page

- View source code in the browser.

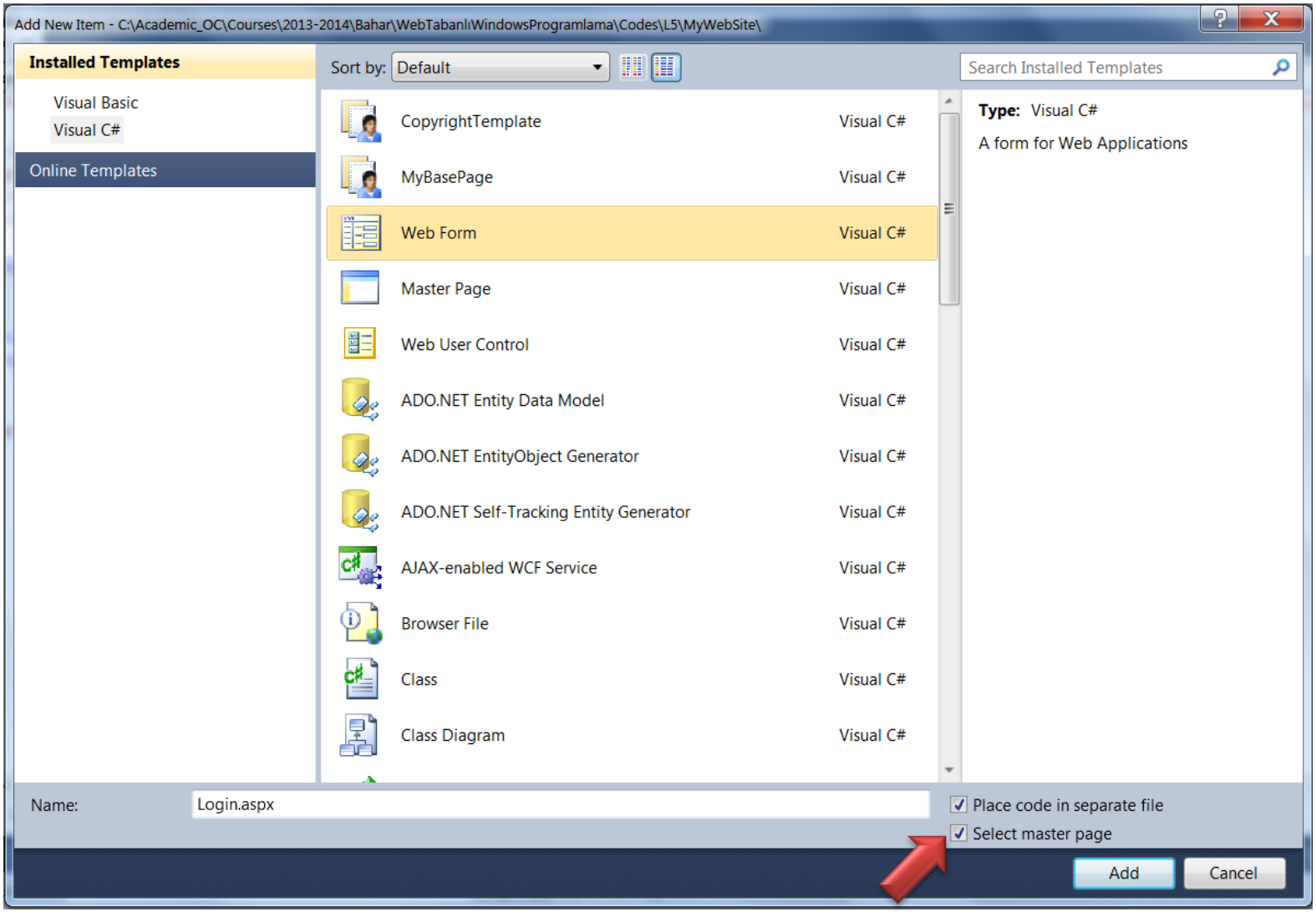
```
<div id="PageWrapper">
    <div id="Header"><a href="./">Header goes here </a></div>
    <div id="MenuWrapper">Menu goes here</div>
    <div id="MainContent">

    <h1 style="padding: 0px; margin: 0px 0px 10px 0px">
        Hi there visitor! Welcome to my Web Site :-)
    </h1>

    <p class="Introduction">
        I&#39;m glad that you are visiting my web site <a href="http://www.mywebsite.com">
        http://www.mywebsite.com</a>
    </p>
    <p class="Introduction">
        <strong>Feel</strong> <span class="style1"><strong>free</strong></span> to have a
        <a href="DefaultWcss2.aspx">look around</a>.
    </p>

    </div>
    <div id="Sidebar">Sidebar goes here</div>
    <div id="Footer">Footer goes here</div>
</div>
```

Example
Adding a Content Page

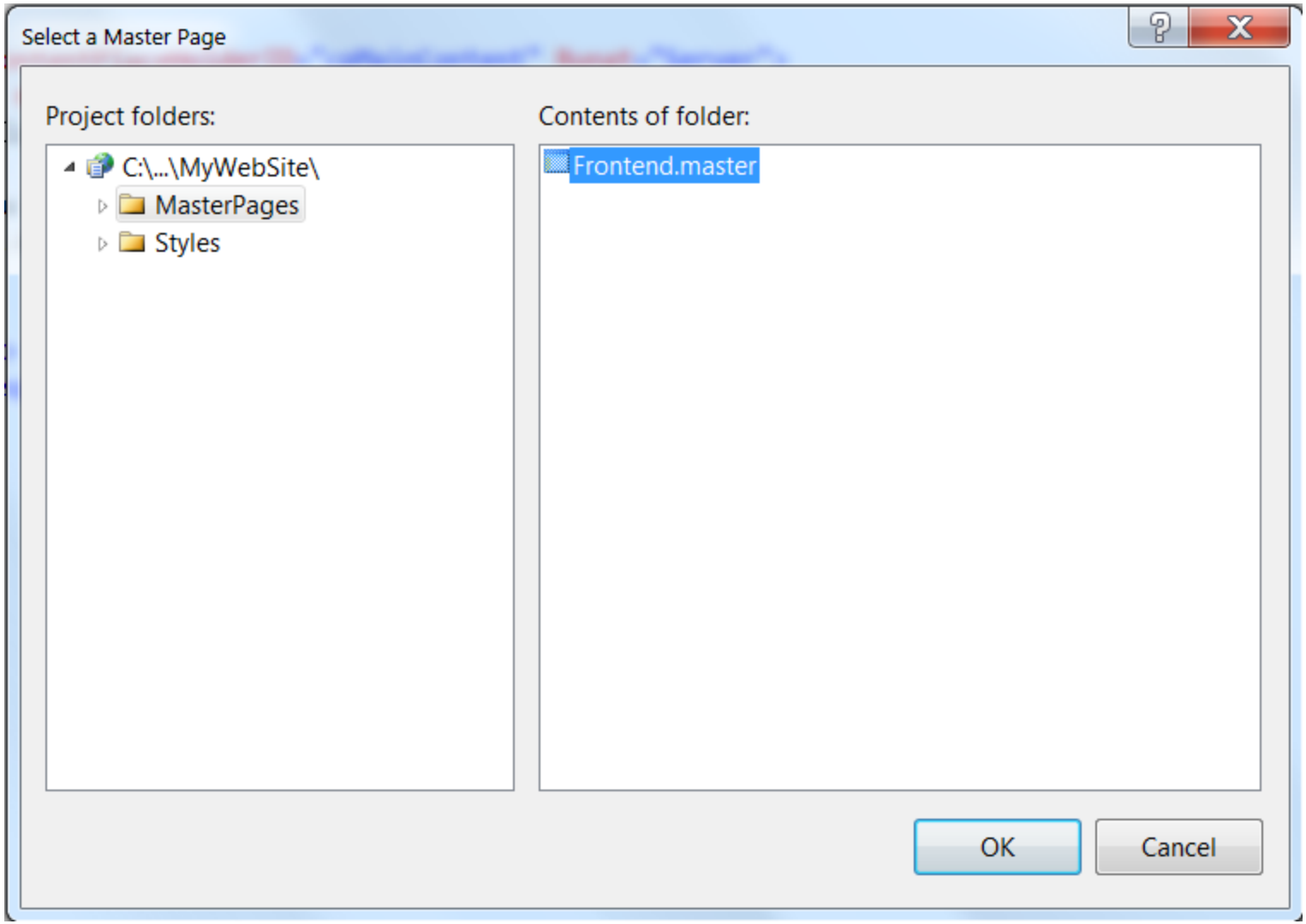
- Switch back to Visual Studio
- Add New Item



Example
Adding a Content Page



Example
Adding a Content Page



Example
Adding a Content Page

- Create an **`<h1>`** element

```
<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
    <h1>Log in to My Website</h1>
</asp:Content>
```

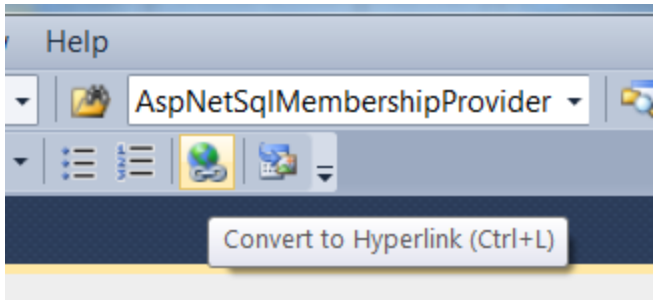
Example
Adding a Content Page

- Go back to **`Default.aspx`**
- Create a new paragraph

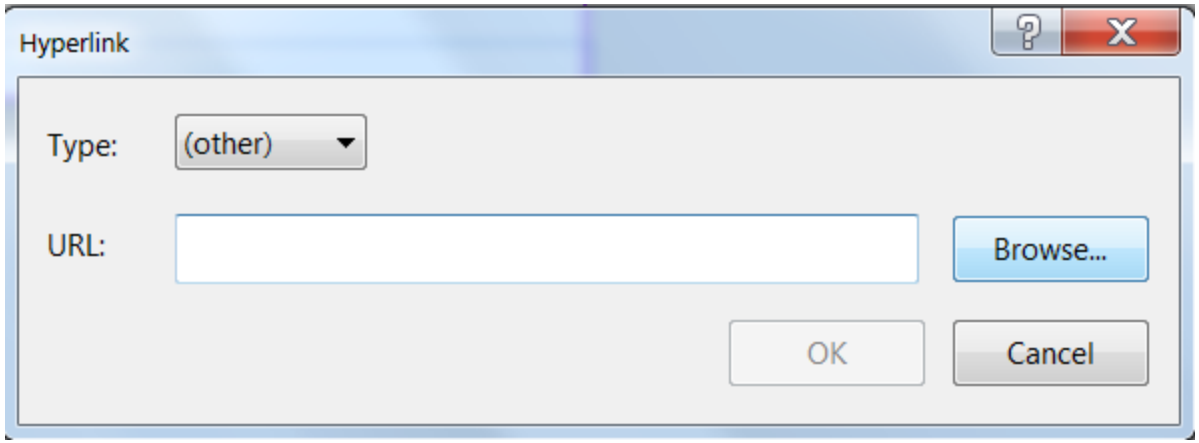
```
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
    <h1 style="padding: 0px; margin: 0px 0px 10px 0px">
        Hi there visitor! Welcome to my Web Site :-)</h1>
    <p class="Introduction">
        I&#39;m glad that you are visiting my web site <a href="http://www.mywebsite.com">
        http://www.mywebsite.com</a>
    </p>
    <p>
        <strong>Feel</strong> <span class="style1"><strong>free</strong></span> to have
        a <a href="Default.aspx">look around</a>.
    </p>
    <p>
        You can log in here
    </p>
</asp:Content>
```

- In Design View → Highlight **log in**

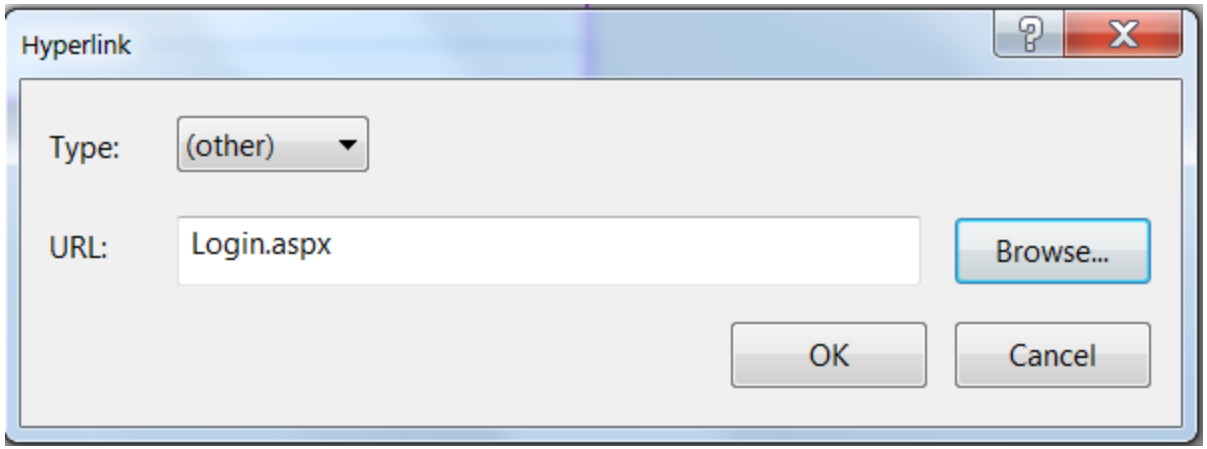
- **Convert to Hyperlink**



- Click **`Browse`**

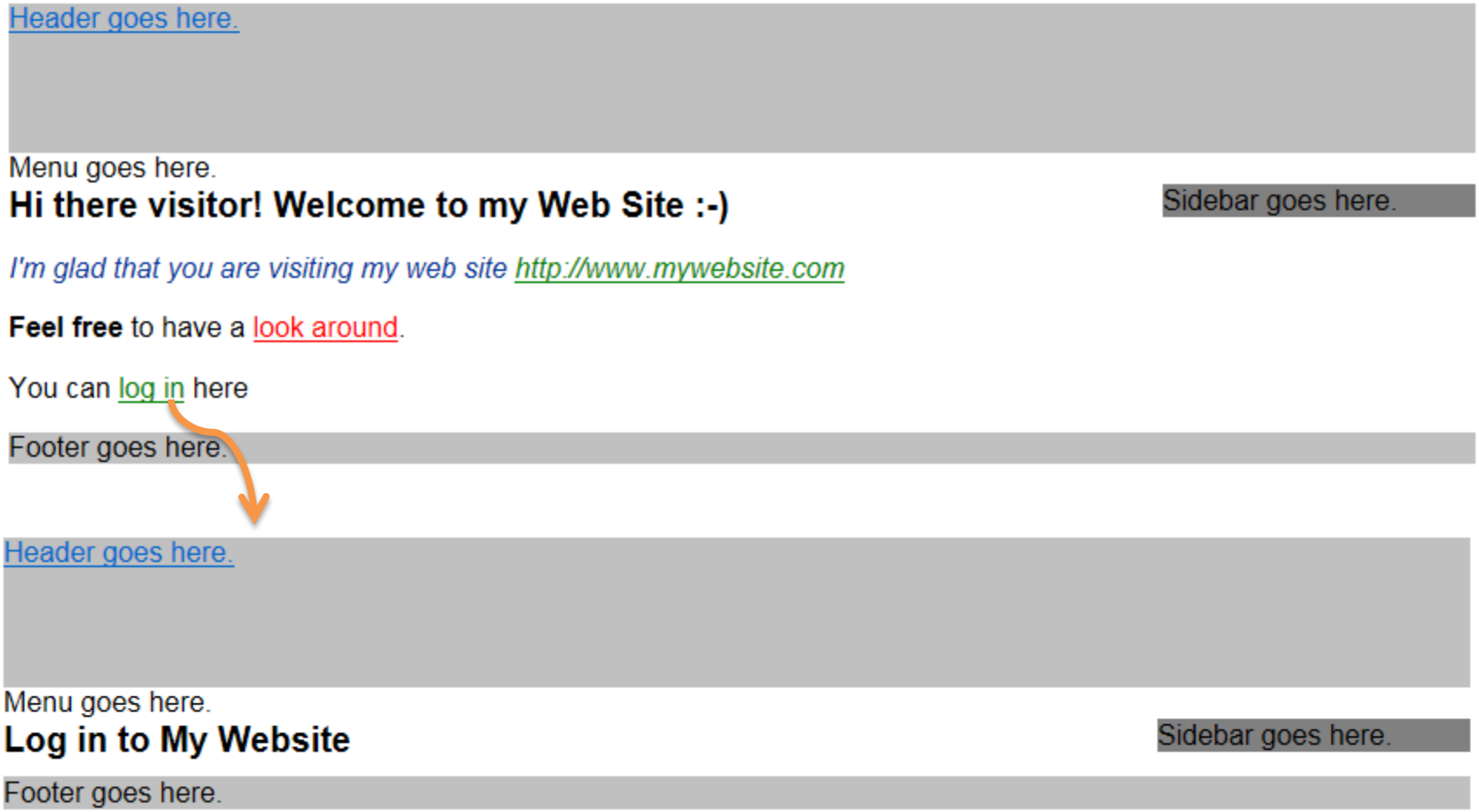


- Select **`Login.aspx`**



Example
Adding a Content Page

- View in browser



Nesting Master Pages

- A nested master page is a **master** that is **based on another master page**.
- Content pages can then be based on the nested master page.
- This is useful if you have a web site that targets different areas that still need to share a common look and feel.
  - For example: A corporate web site that is separated by departments.

Master Page Caveats

- In a normal ASPX page:

```
<asp:Button ID="Button1" runat="server" Text="Click Me" />
```

- In the final HTML, ends up with:

```
<input type="submit" name="Button1" value="Click Me" id="Button1" />
```

- However, inside an `<asp:Content>` control:

Auto generated `ID` of the master page

```
<input type="submit" name="ctl00$cpMainContent$Button1" value="Click Me"
                id="cpMainContent_Button1" />
```

ID of the `ContentPlaceHolder` control

Master Page Caveats

- Because the **`name`** and **`id`** of the HTML elements are changed, they add **considerably to the size of the page**.
  - Gets worse **with nested master pages**:

```
<input type="submit" name="ctl00$ctl00$cpMainContent$ContentPlaceHolder1$Button1"
      value="Click Me" id="cpMainContent_ContentPlaceHolder1_Button1" />
```

- You should keep the IDs of your `ContentPlaceHolder` and `Content` controls **as short as possible**.

Master Page Caveats

- Master page <u>improves</u> **the consistency** and **maintability of your site**.

- **Base Page** → Another way to improve consistency
  - Centralize the behavior of the pages in your web site.

Base Page

- By default, pages in your web site inherit from the **`Page`** class defined in the **`System.Web.UI`** namespace.
  - However, in some circumstances this behavior is not enough and you need to add your own stuff.
- You can easily insert your own **`base page`** class between a web page and the standard **`Page`** class.

Base Page

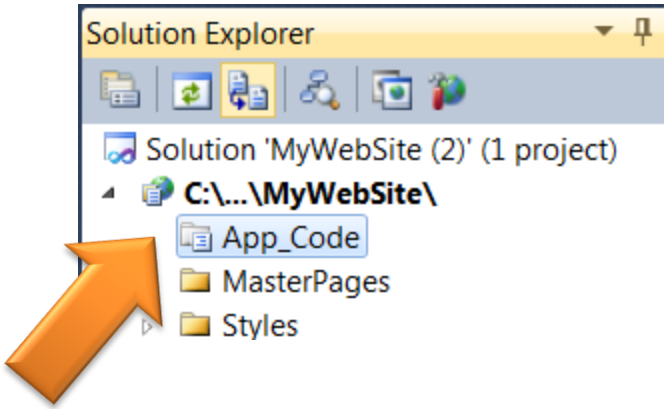
- To have your pages inherit from this base page, you need to do two things:

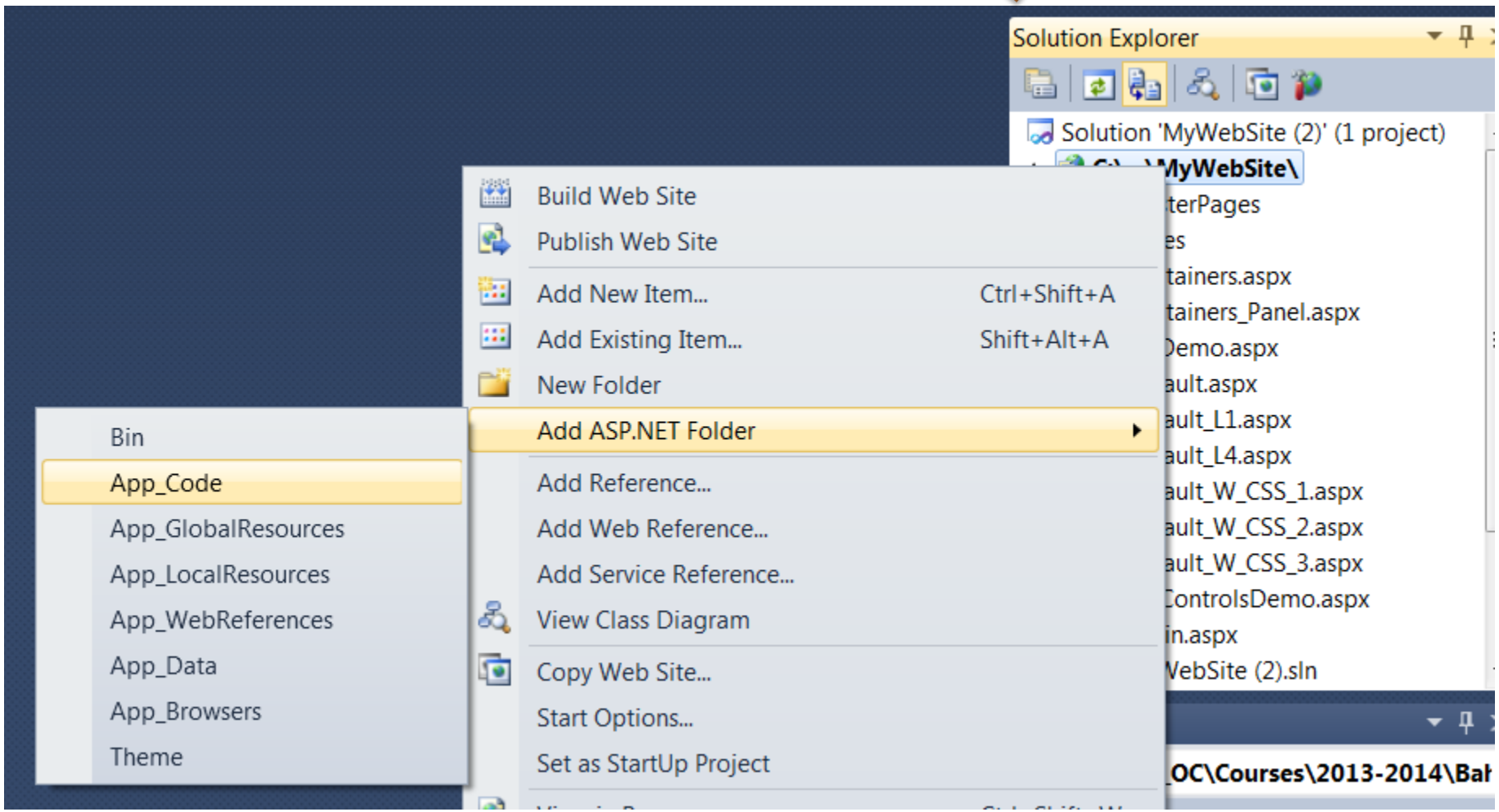
  1. **Create a class** that inherits from **`System.Web.UI.Page`** in the **`App_Code`** folder of your web site.

  2. **Make the web pages in your site inherit from this base page** instead of the standard **`Page`** class.

Example Base Page

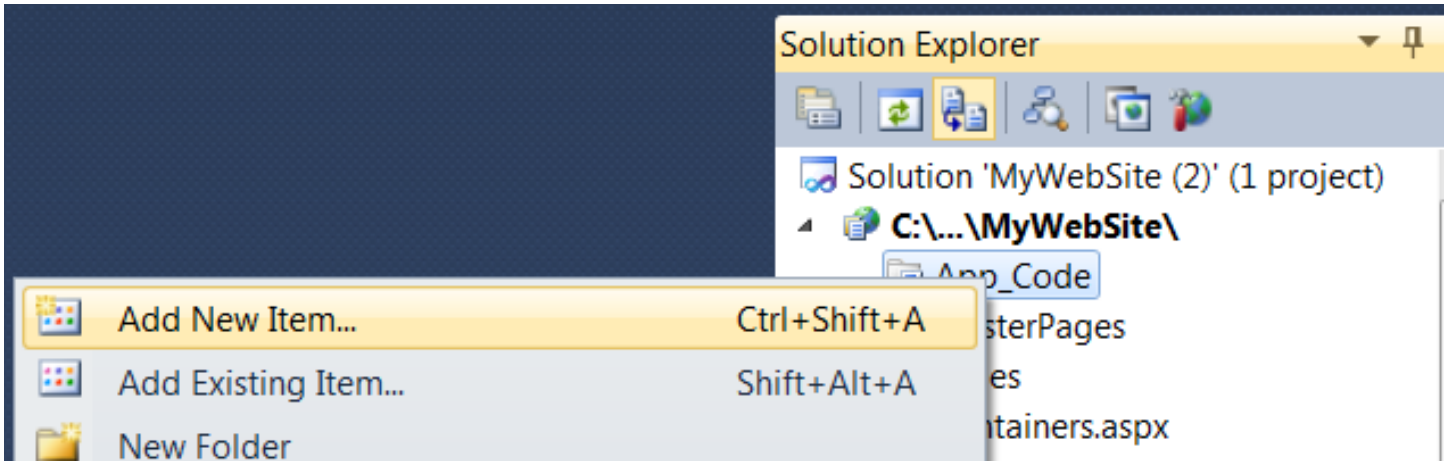
- Add **App_Code** folder



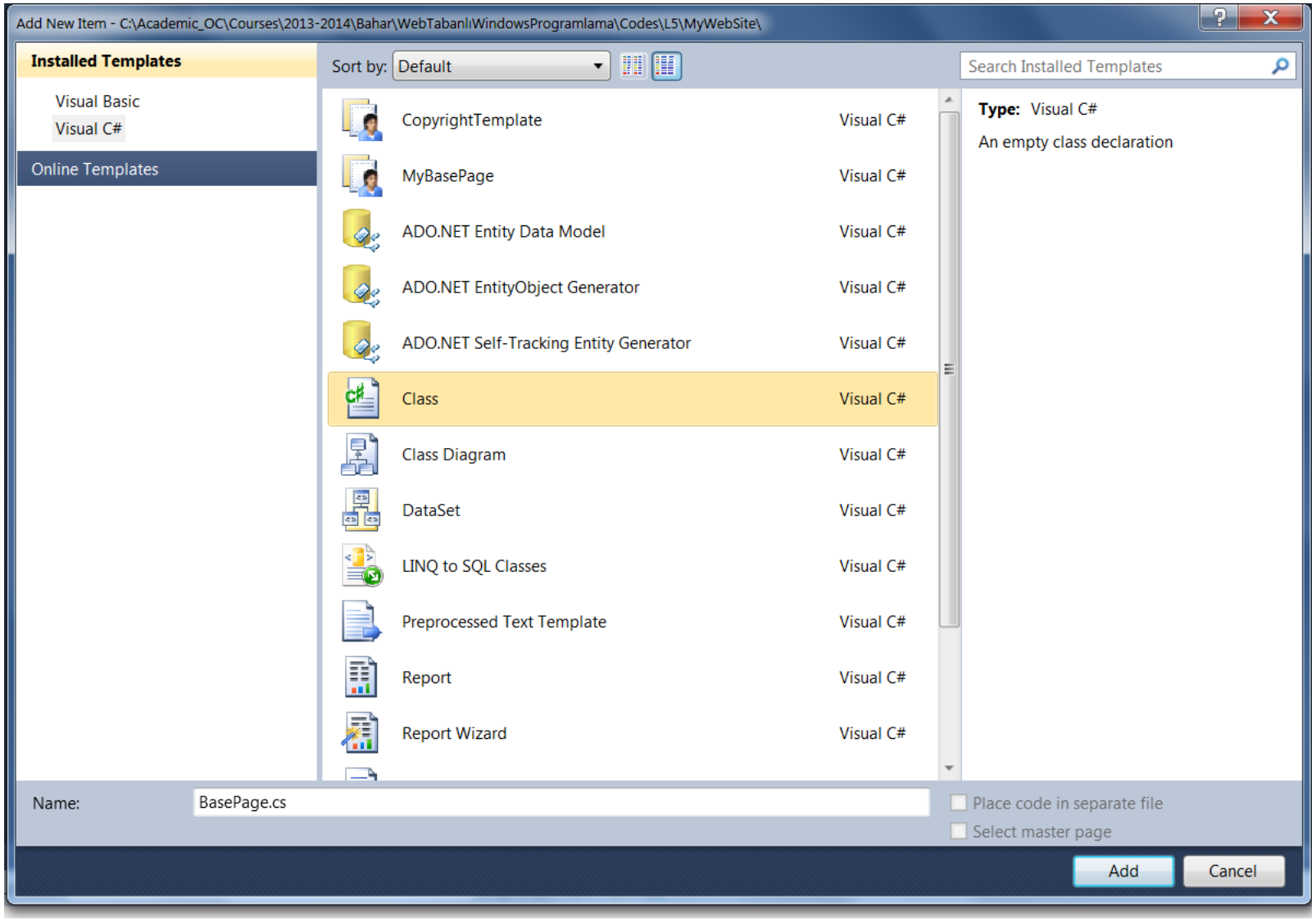


Example
Base Page

- Add New Item to **`App_Code`** folder



Example
Base Page



Example
Base Page

```
public class BasePage:System.Web.UI.Page
{

    private void Page_PreRender(object sender, EventArgs e)
    {
        if (this.Title == "Untitled Page" || string.IsNullOrEmpty(this.Title))
        {
            throw new Exception("Page title cannot be \"Untitled Page\" or an empty string.");
        }
}

    public BasePage()
    {
        //
        // TODO: Add constructor logic here
        //

        this.PreRender += new EventHandler(Page_PreRender);
    }
}
```

Example
Base Page

- Open Code Behind file of **`Login.aspx`**

```
public partial class Login : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
}
```

```
public partial class Login : BasePage
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
}
```

Example
Base Page

- View **`Login.aspx`** in browser

Server Error in '/MyWebSite' Application.

Page title cannot be "Untitled Page" or an empty string.

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.Exception: Page title cannot be "Untitled Page" or an empty string.

Source Error:

```
The source code that generated this unhandled exception can only be shown when compiled in debug mode. To enable this, please
follow one of the below steps, then request the URL:

1. Add a "Debug=true" directive at the top of the file that generated the error. Example:

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

or:

2) Add the following section to the configuration file of your application:

<configuration>
   <system.web>
       <compilation debug="true"/>
   </system.web>
</configuration>
```

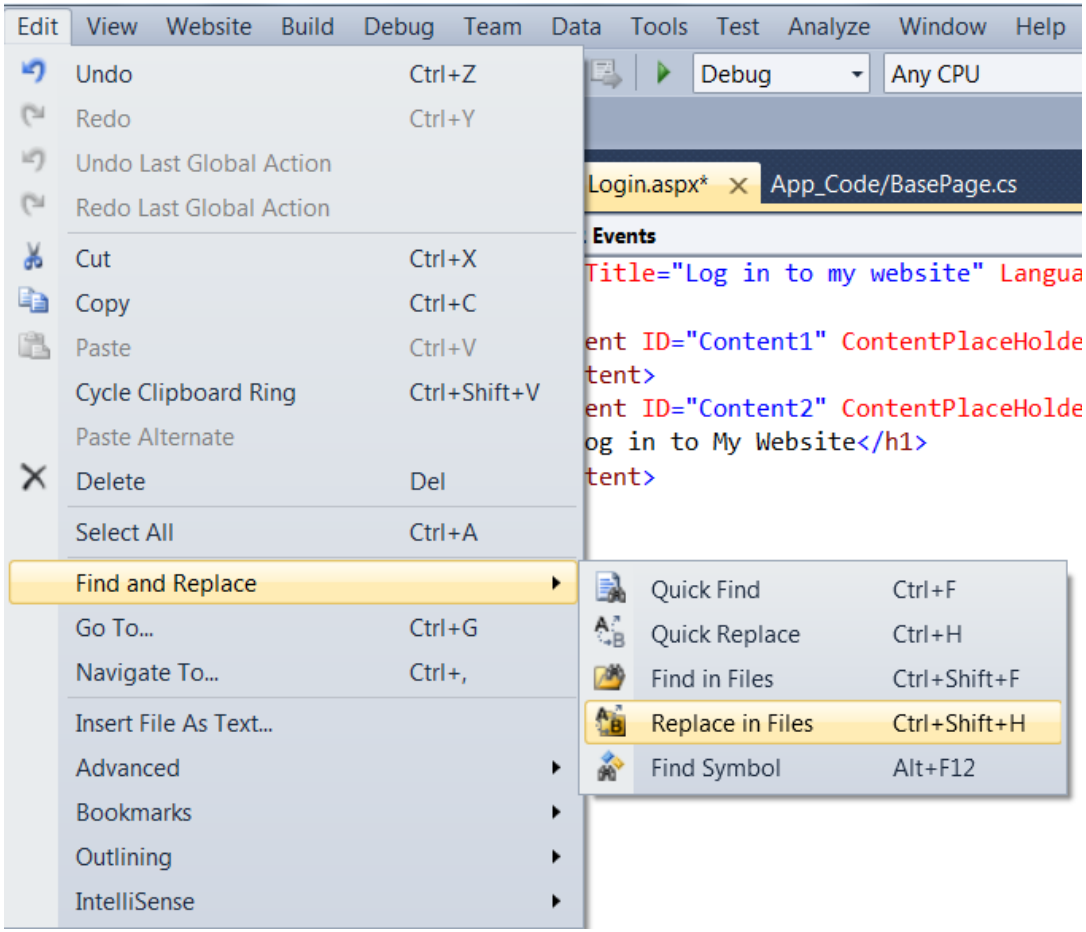
Example
Base Page

- Go back to Visual Studio
- Set value to **`Title`** attribute in the **`@Page`** directive

```
<%@ Page Title="Log in to my website" Language="C#" MasterPageFile="~/MasterPages/Frontend.master"
```

Example
Base Page

- Make a replacement for the rest of the files



Find and Replace

Quick Find | Replace in Files

Find what:

System.Web.UI.Page

Replace with:

BasePage

Look in:

Entire Solution

Include sub-folders

Find options

Match case

Match whole word

Use:

Regular expressions

Look at these file types:

*.aspx.cs

Result options

Find Next | Replace

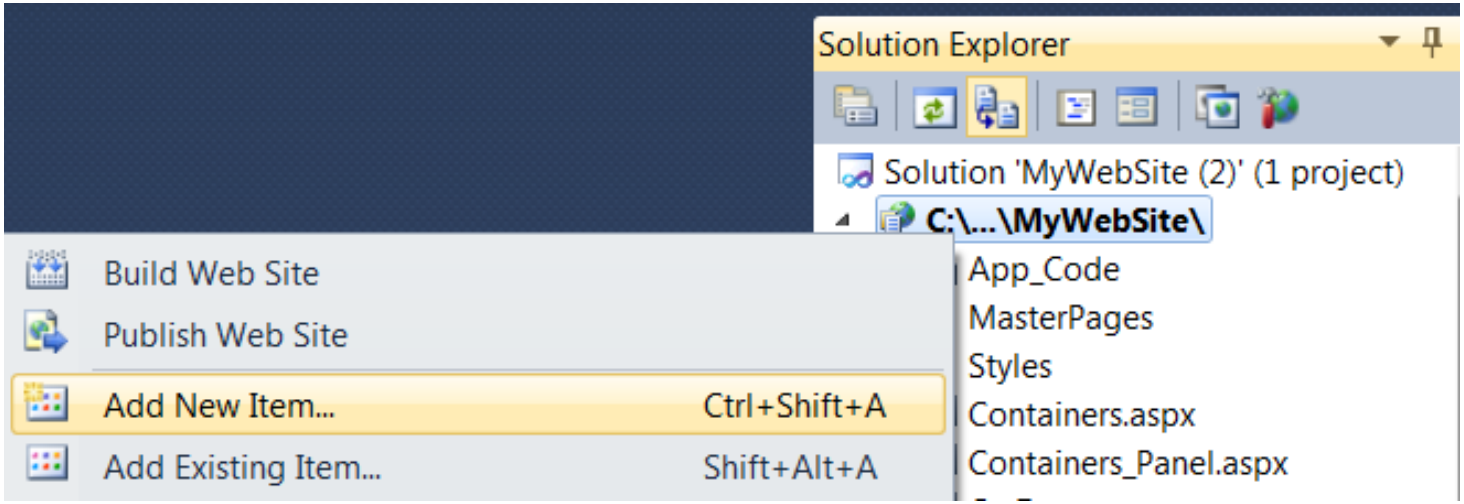
Skip File | Replace All

Example
Base Page

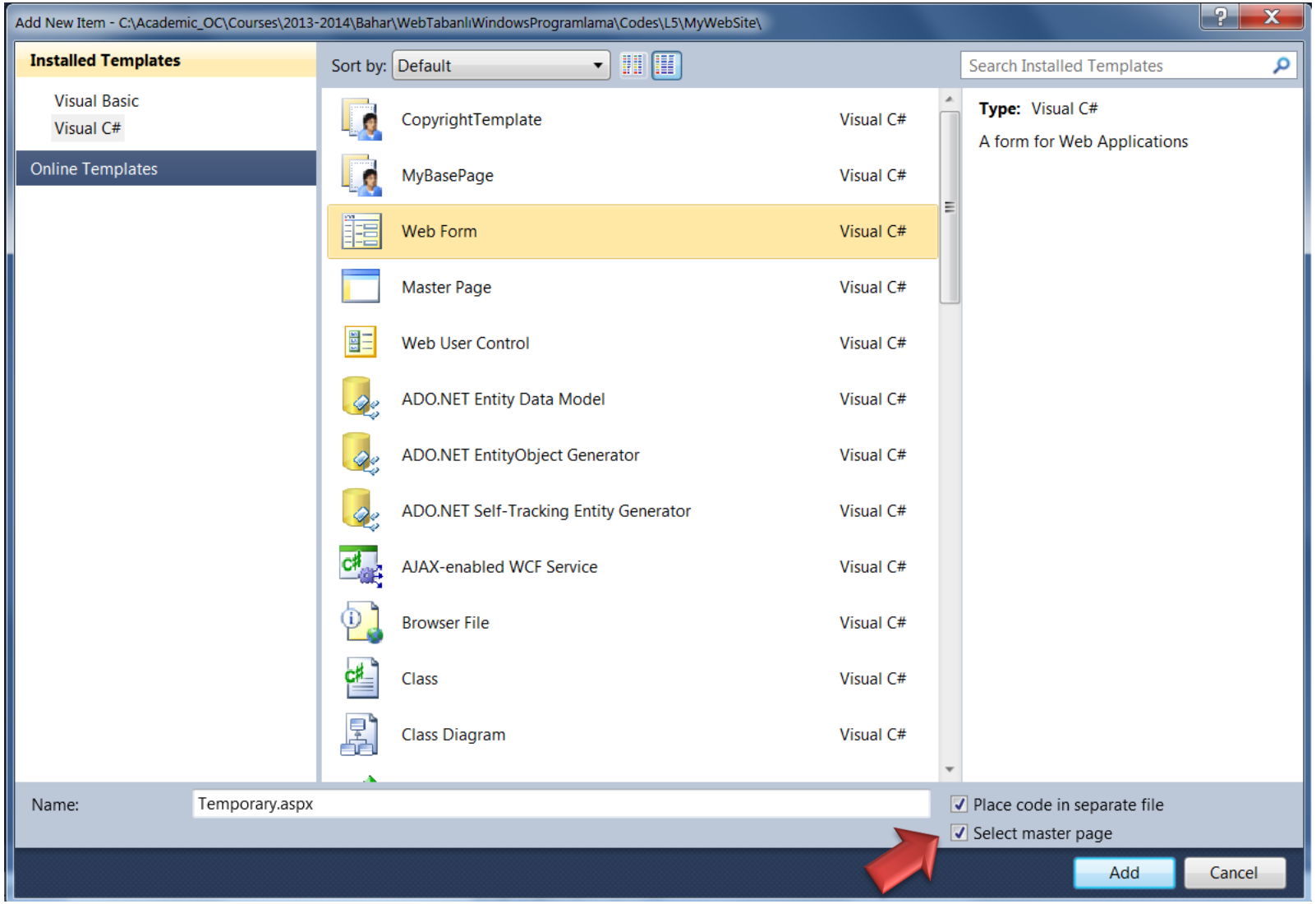
- View Login.aspx in browser

Example
Reusable Page Template

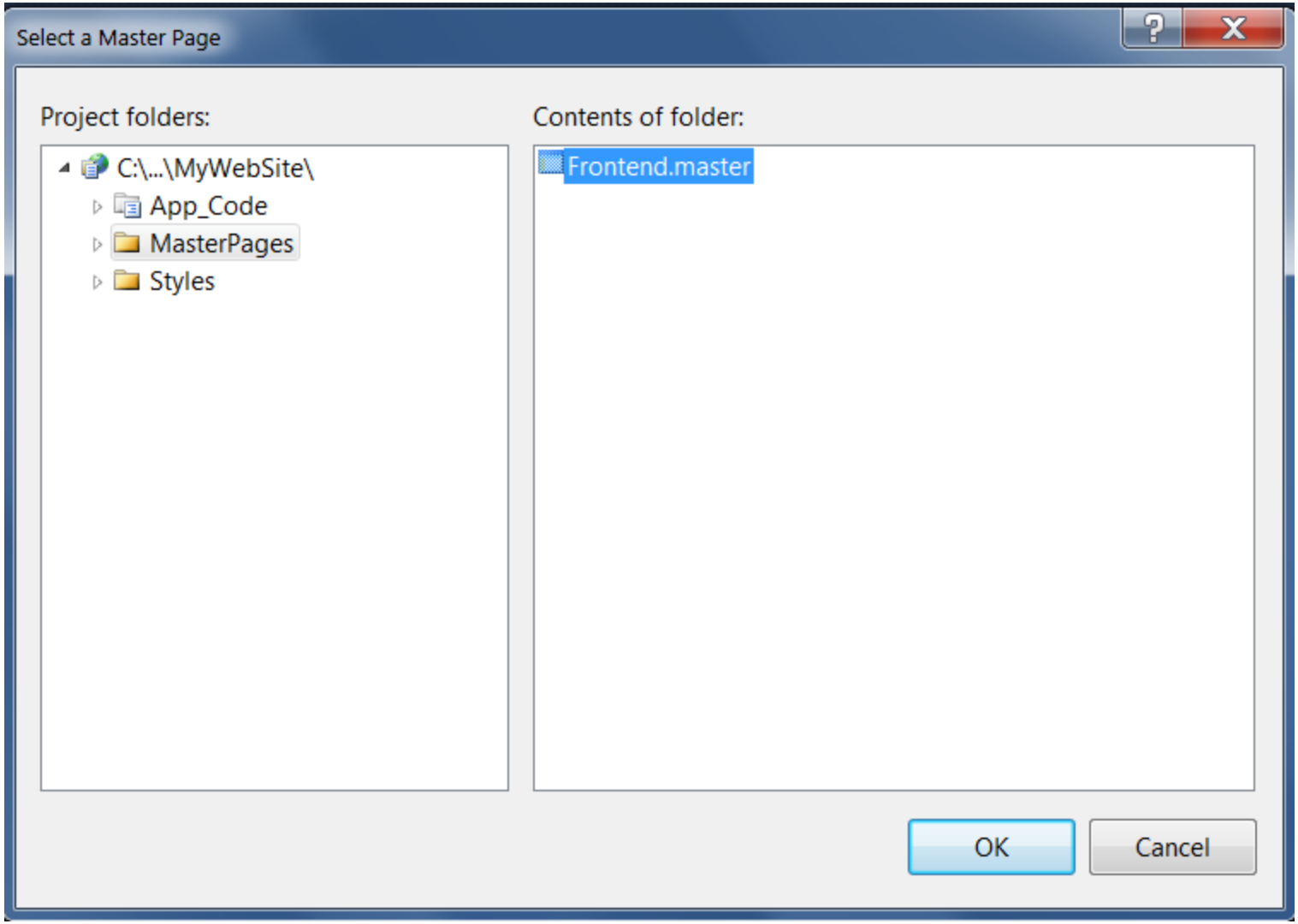
- Add New Item



Example
Reusable Page Template



Example
Reusable Page Template



Example
Reusable Page Template

- Open Code Behind file

```
public partial class Temporary : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
}
```

Name of the folder

Name of the page

```
public partial class $relurlnamespace$_$safeitemname$ : BasePage
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
}
```

Example
Reusable Page Template

- In Source View → Change the **`Inherits`** attribute

```
<%@ Page Title="" Language="C#" MasterPageFile="~/MasterPages/Frontend.master" AutoEventWireup="true"
                                                    CodeFile="Temporary.aspx.cs" Inherits="Temporary" %>
```

```
<%@ Page Title="" Language="C#" MasterPageFile="~/MasterPages/Frontend.master" AutoEventWireup="true"
                                                    CodeFile="Temporary.aspx.cs" Inherits="$relurlnamespace$_$safeitemname$" %>
```

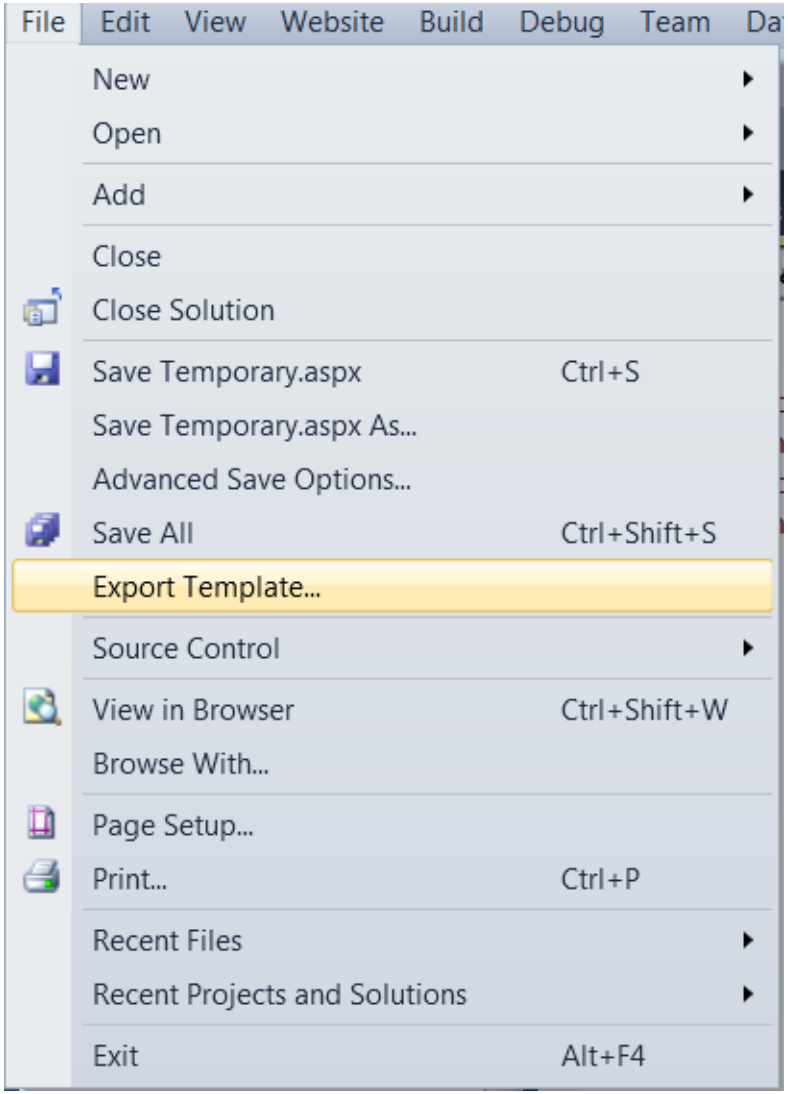
Example
Reusable Page Template

- Add a comment about copyright notice.

```
<%@ Page Title="" Language="C#" MasterPageFile="~/MasterPages/Frontend.master" Au

<%--Please email to me@mywebsite.com for copyright.--%>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
</asp:Content>
```

Example
Reusable Page Template

- Save all changes



Export Template Wizard

Choose Template Type

This wizard will allow you to export a project or project item from the current solution to a template which future projects can then be based upon.

Which type of template would you like to create?

- Project template

  A project template will allow a user to create a new project based on your exported project. A user will be able to utilize your template from the New Project dialog box for client projects and from the New Website dialog box for websites.

- Item template

  An item template will allow a user to add your item to one of their existing project. Your template will be available to the user from the Add New Item dialog box.

From which project would you like to create a template?

MyWebSite

What language category should this template appear under in the New Project dialog box?

Visual C#

< Previous | Next > | Finish | Cancel

Export Template Wizard

Select Item To Export

Select the item that you would like to export as an item template. All dependent files (including designer and resource files) will automatically be included with the selected item in the exported template.

Item to export:

- [ ] Default_W_CSS_2.aspx
- [ ] Default_W_CSS_3.aspx
- [ ] ListControlsDemo.aspx
- [ ] Login.aspx
- [ ] Login_V1.aspx
- [ ] MasterPages
  - [ ] Frontend.master
- [ ] MyWebSite (2).sln
- [ ] ServerControlsDemo.aspx
- [ ] State.aspx
- [ ] State_1.aspx
- [ ] Styles
  - [ ] Styles.css
- [x] Temporary.aspx
- [ ] ViewState.aspx
- [ ] web.config

< Previous | Next > | Finish | Cancel

Export Template Wizard

Select Item References

Select the references you would like to include with this item:

- System.Web.Services, Version=4.0.0.0, Culture=neutral, PublicKeyToken=B03F5F7F11D50A3A
- System.ServiceModel.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35
- System.IdentityModel, Version=4.0.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089
- System.WorkflowServices, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35
- System.Core, Version=4.0.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089
- System.Activities, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35
- System.ComponentModel.DataAnnotations, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31BF3856
- System.Xml.Linq, Version=4.0.0.0, Culture=neutral, PublicKeyToken=B77A5C561934E089

< Previous | Next > | Finish | Cancel

Export Template Wizard

Select Template Options

Template name:

MyWebSiteBasePage

Template description:

Used in my website to create pages that are based on a central BasePage and a MasterPage

Icon Image:

Browse...

Preview Image:

Browse...

Output location:

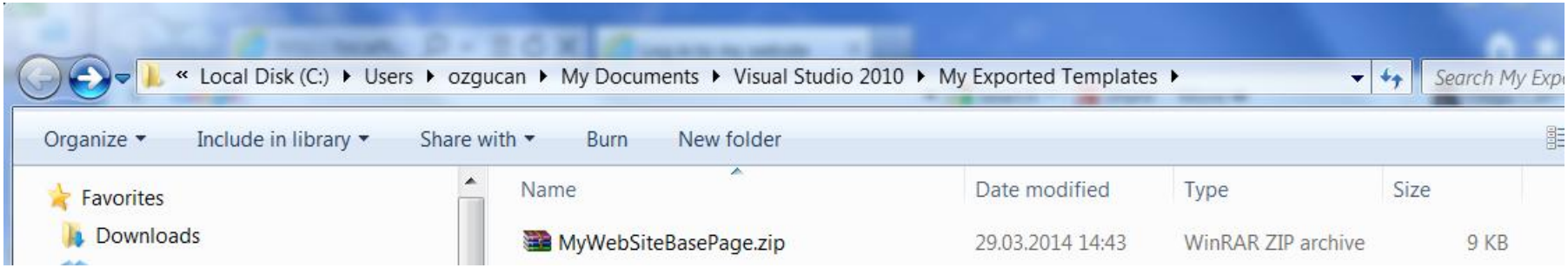
C:\Users\ozgucan\Documents\Visual Studio 2010\My Exported Templates\MyWebSiteBasePage.zip

☑ Automatically import the template into Visual Studio

☑ Display an explorer window on the output files folder

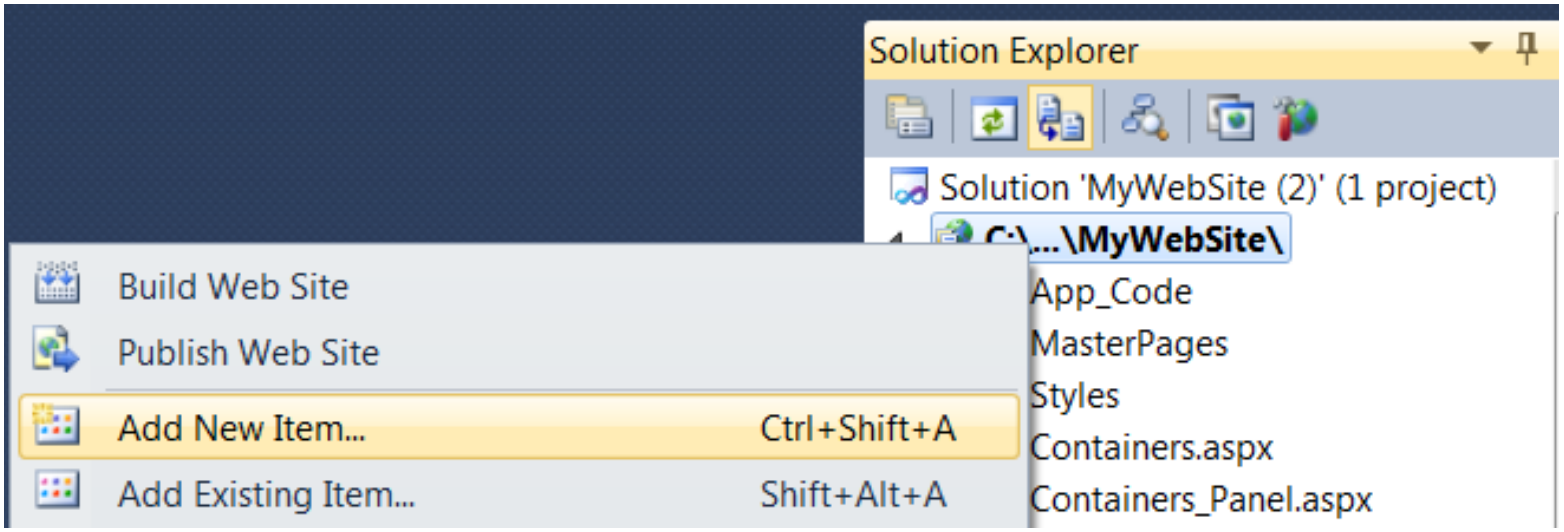
< Previous | Next > | Finish | Cancel

Example
Reusable Page Template

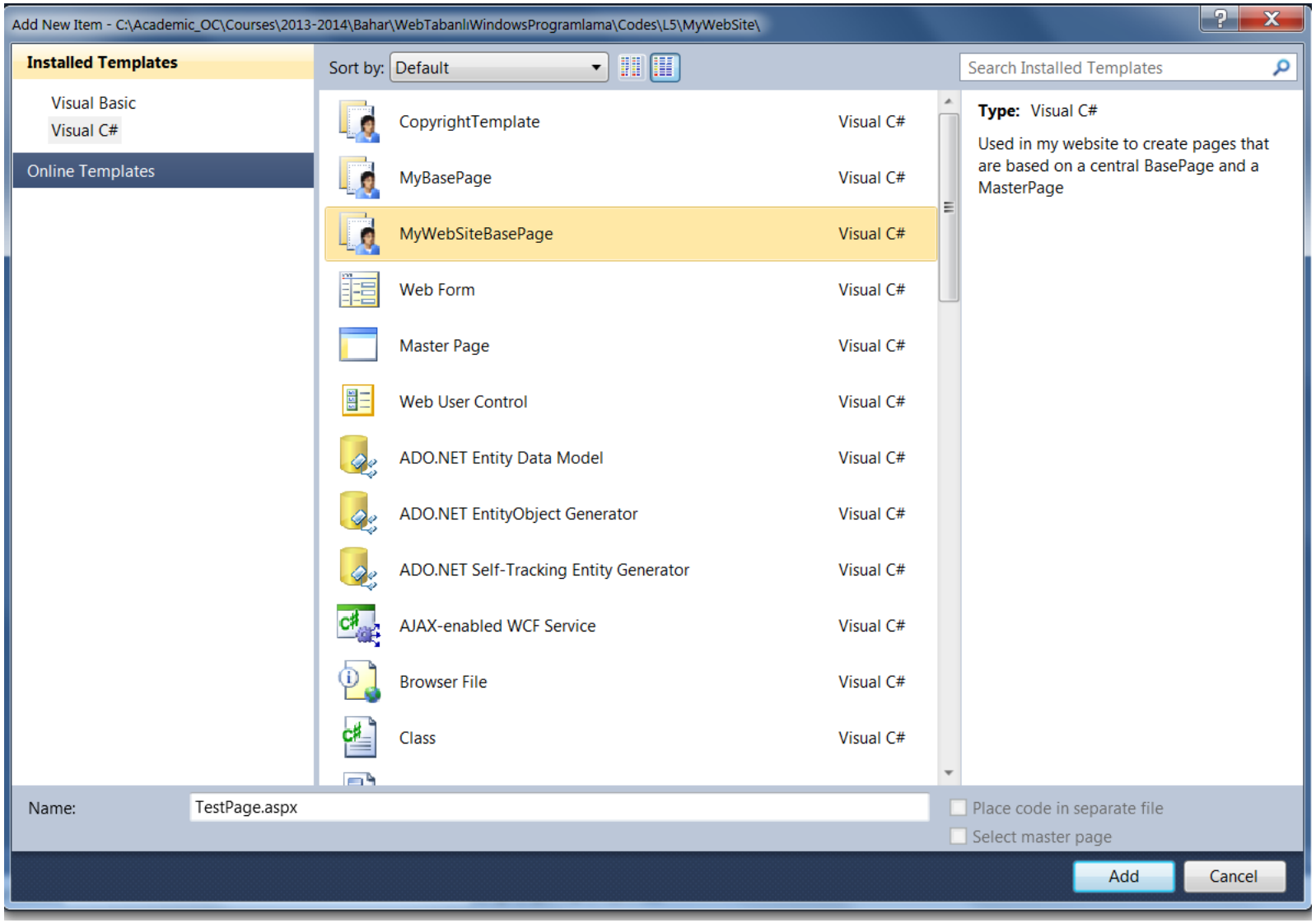


Example
Reusable Page Template

- Delete **`Temporary.aspx`**
- Add New Item



Example
Reusable Page Template



Example
Reusable Page Template

- Check out the source file and Code Behind file

TestPage.aspx*

Client Objects & Events | (No Events)

```
<%@ Page Title="" Language="C#" MasterPageFile="~/MasterPages/Frontend.master" AutoEventWireup="true"
            CodeFile="TestPage.aspx.cs" Inherits="_TestPage" %>

<%--Please email to me@mywebsite.com for copyright.--%>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
</asp:Content>
```

```
public partial class _TestPage : BasePage
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
}
```

Page Life Cycle

How a page is served by a web server to the browser → ***Life cycle of a page***

1. Page Request
2. Start
3. Page Initialization
4. Load
5. Validation
6. Postback Event Handling
7. Rendering
8. Unload

Please READ
http://msdn.microsoft.com/en-us/library/ms178472(v=vs.100).aspx

PHASE	DESCRIPTION
Page request	A request to an ASPX page starts the life cycle of that page. When the web server is able and allowed to return a cached copy of the page, the entire life cycle is not executed. In all other situations, the page enters the start phase.
Start	In this phase, the page gets access to properties like `Request` and `Response` that are used to interact with the page's environment. In addition, during this phase the `PreInit` event is raised to signal that the page is about to go into the initialization phase.
Page initialization	During this phase, the controls you have set up in your page or added programmatically become available. Additionally, the `Page` class fires three events: `Init`, `InitComplete`, and `PreLoad`. Also during this phase, the control properties are loaded from View State and Control State again during a postback. So, for example, when you change the selected item in a `DropDownList` and then cause a postback, this is the moment where the correct item gets preselected in the drop-down list again, which you can then work with in your server-side code.
Load	During this phase the page raises the `Load` event.
Validation	In the validation phase, the Validation controls used to validate user input are processed.
Postback event handling	During this phase, the controls in your page may raise their own events. For example, the `DropDownList` may raise a `SelectedIndexChanged` event when the user has chosen a different option in the list. Similarly, a `TextBox` may raise the `TextChanged` event when the user has changed the text before she posted back to the server. When all event processing is done, the page raises the `LoadComplete` event. During this phase the `PreRender` event is raised to signal that the page is about to render to the browser. Shortly after that, `SaveStateComplete` is raised to indicate that the page is done storing all the relevant data for the controls in View State.
Rendering	Rendering is the phase where the controls (and the page itself) output their HTML to the browser.
Unload	The unload phase is really a clean-up phase. This is the moment where the page and controls can release resources like database connections. During this phase, the `Unload` event is raised so you can handle any cleanup you may need to do.