Introduction to ASP.NET Web Development
Instructor: Frank Stepanski

Overview

From this class, you will learn how to develop web applications using the Microsoft web technology ASP.NET using the free web development tool Visual Studio 2012 Express.

Visual Studio 2012 Express for Web will allow you to create web pages (.aspx) and web sites that use the .NET framework and the specific objects and controls of ASP.NET.

Free download:
http://www.microsoft.com/visualstudio/eng/products/visual-studio-express-for-web

Note: If you already have an older version of the free or commercial tool, that will be fine for this beginner class.

ASP.NET File Extensions

.aspx

When you go to a web browser and the web page has this extension, you know it is an ASP.NET page.

An additional extension would be **.aspx.cs**. This is the file extension where all your C# code will be stored ("code-behind"). You never see this extension on the web because this file gets compiled on the web server that contains the .NET Framework.
Note: The **.cs** stands for C# (# is "sharp").

If you use a different programming language for ASP.NET pages like VB.NET, it would have an **.aspx.vb** extension instead.

Six Important Facts about ASP.NET

1. ASP.NET is integrated within the .NET Framework:

The .NET Framework is divided into an almost painstaking collection of functional parts, with a staggering total of more than 8000 types (functions, classes, interfaces, etc.). The .NET framework allows for the development of Windows and Web applications. This means ASP.NET is only a part of the entire framework of .NET.

2. ASP.NET is Compiled, Not Interpreted:

ASP.NET applications actually go through two stages of compilation. In the first stage, the C# code you write (or whatever .NET language it's written in) is compiled into an intermediate language called Microsoft Intermediate Language (MSIL). The second level compilation happens just before the page is actually executed. At this point, the IL code is compiled into low-level native machine code. This stage is known as just-in-time (JIT) compilation and it takes place in the same way for all .NET applications (including Windows applications).

3. ASP.NET is Multi-language:

Though you will probably choose one language over another, you can use any .NET supported programming language to create your ASP.NET web pages. You can even use a different programming language for each new ASP.NET web page, although you probably don't want to. The default supported and most popular languages for ASP.NET are C# and VB.NET.

4. ASP.NET is Object-Oriented:

Not only does your code have full access to all objects in the .NET Framework, but you can also exploit all the conventions of OOP (object-oriented programming) environment, such as encapsulation and inheritance. For example, you can create reusable classes, standardize code with interfaces and bundle useful functionality in a distributable, compiled component.

5. ASP.NET is Easy to Deploy and Configure:

Every installation of the .NET Framework provides the same core classes. As a result, deploying an ASP.NET application is relatively simple. In most cases, you simply need to copy all the files to a directory on a web server (web host). There are configuration settings that can be made in a **web.config** file.

6. ASP.NET is Multidevice and Multibrowser:

One of the greatest challenges web developers face is the wide variety of browsers they need to support. Different browser brands, versions, and configurations differ in their support of XHTML. ASP.NET server controls render their HTML adaptively by taking the client's capabilities into account.

Web Server for ASP.NET

To view your ASP.NET web pages, it needs to be requested through a web browser running on a web server. The web server then dispatches the request to the ASP.NET engine installed on the web server. The ASP.NET engine processes the page and returns the resulting HTML markup to the browser.

ASP.NET web pages are normally developed and tested locally on your PC. You could develop them locally and test them on your webhost (FTP-ing the files over each time).

Visual Web Developer includes a built-in web server. So anytime you want to test your **.aspx** web pages that you've developed, all you have to do is launch a web browser from Visual Web Developer and it will create a running instance of a temporary web server.

Note: If you have any "professional" version of the Windows operating system, you have a Microsoft web server (IIS – Internet Information Services) already. You can use this web server instead if you want.

Brief Tour of Visual Studio 2012 for Web

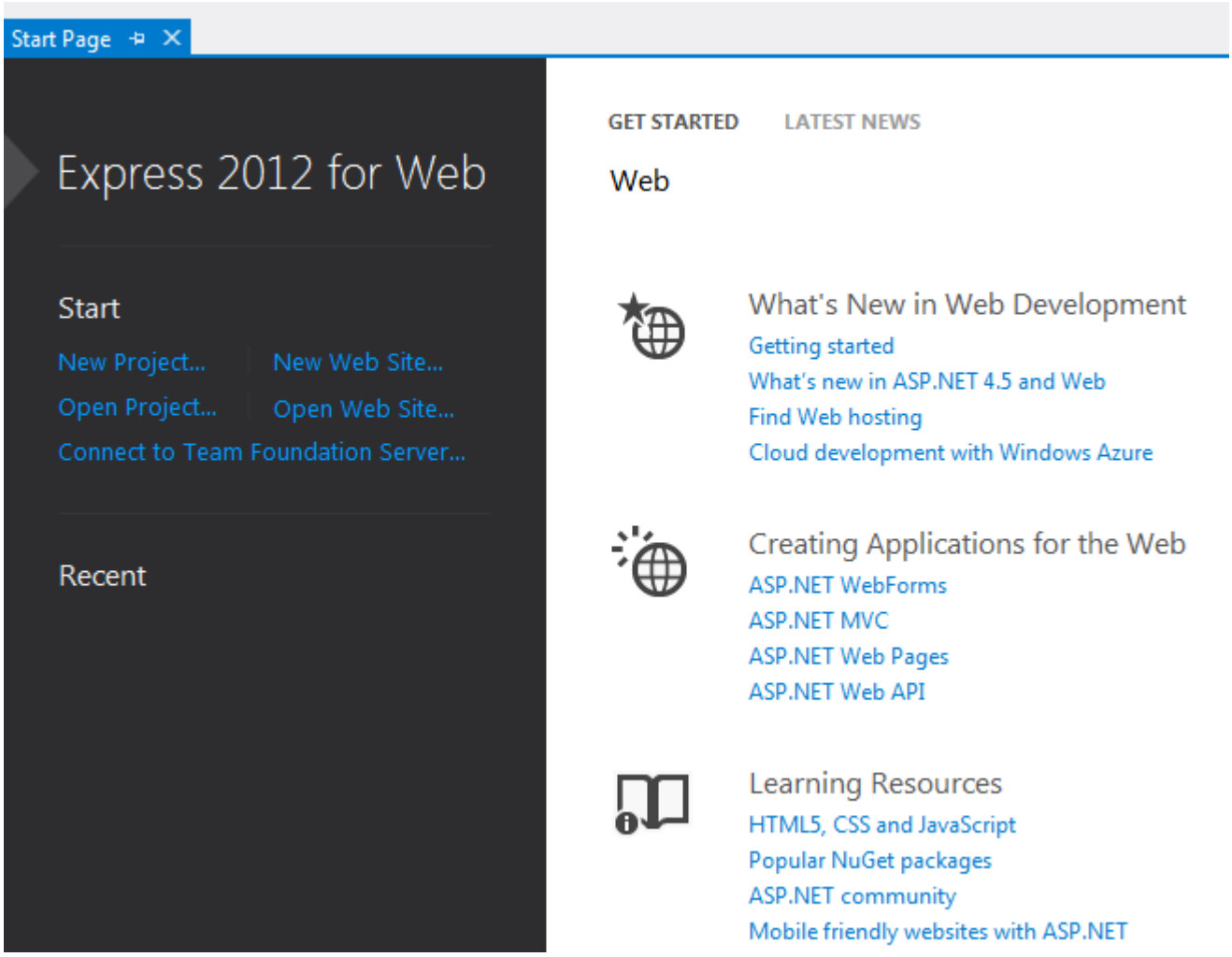


Figure 1 – Visual Studio Express 2012 for Web

When you open Visual Studio 2012, the Start Page is initially shown (Figure 1).

This Start Page includes a list of Recent Projects in the upper-left corner, a Getting Started section with some links for accomplishing common tasks in the bottom left-corner, and a list of recent articles on Microsoft's MSDN site in the right column.

On the left you'll find the Toolbox. On the Start Page, the Toolbox is empty, but when you're working with an ASP.NET page, the Toolbox contains the plethora of ASP.NET Web controls that can be added to the page.

To the right of the screen, you'll find the Solution Explorer. Again, on the Start Page this is empty, but when you load or create an ASP.NET website, the Solution Explorer will list the website's files. These files include database files, XHTML pages, ASP.NET pages, image files, CSS files, configuration files, and so on. In addition to the Solution Explorer, the right portion of the screen is also home to the Database Explorer.

Note: If you accidentally close the Solution Explorer, you can re-open it by going to View->Solution Explorer from the menu.

Creating a New ASP.NET Website

To create and work with an ASP.NET page, we must first create an Empty ASP.NET website.

Note: We are creating an 'Empty' ASP.NET website because Visual Studio Express 2012 adds files to a vase website that I don't you to worry about this early in the class.

You can go to the File menu and choose the New Web Site option and choose Empty Web ASP.NET Website option (Figure 2b).

Make sure you choose the C# language (on the left) as the language for all the web pages (Figure 2b). You can also specify the location of the folder where all your website files will be stored at the bottom (web location – file system).

Normally, you will create a new folder location for each new website.

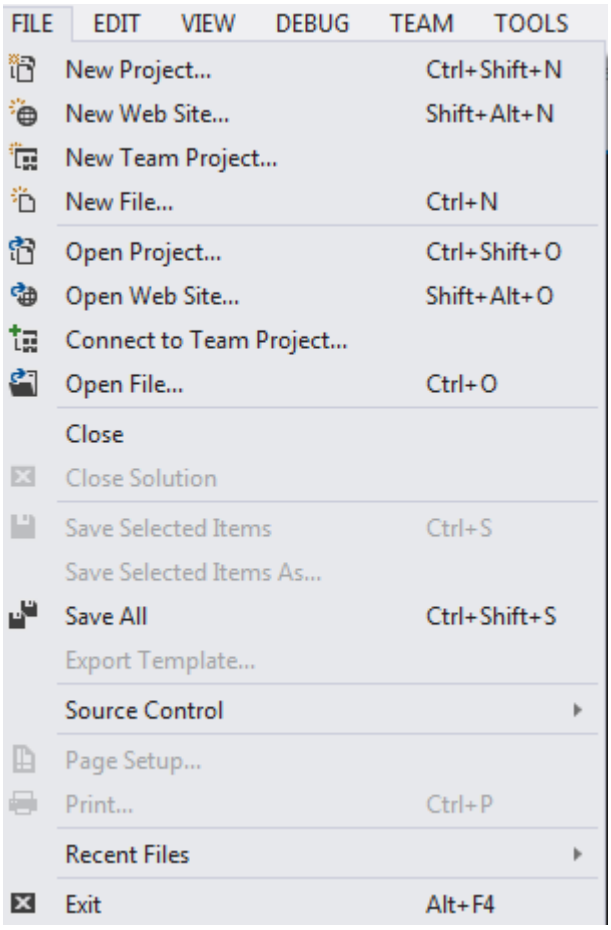


Figure 2a – Creating a New ASP.NET Website

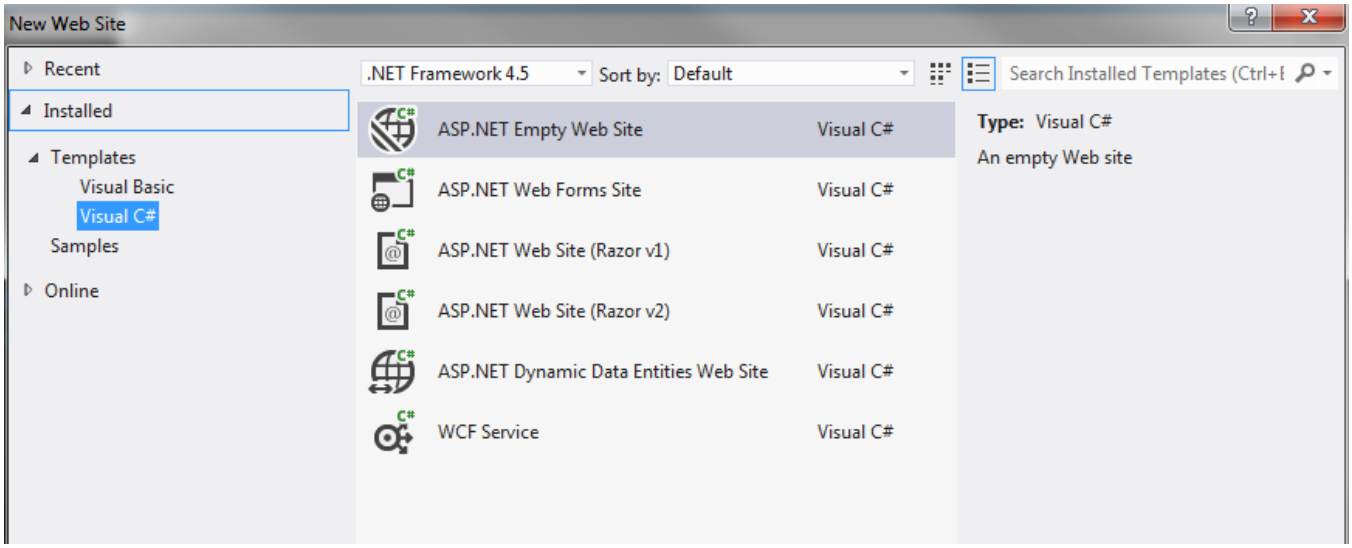


Figure 2b – Creating a New ASP.NET Website: Empty Web Site (select Visual C#)

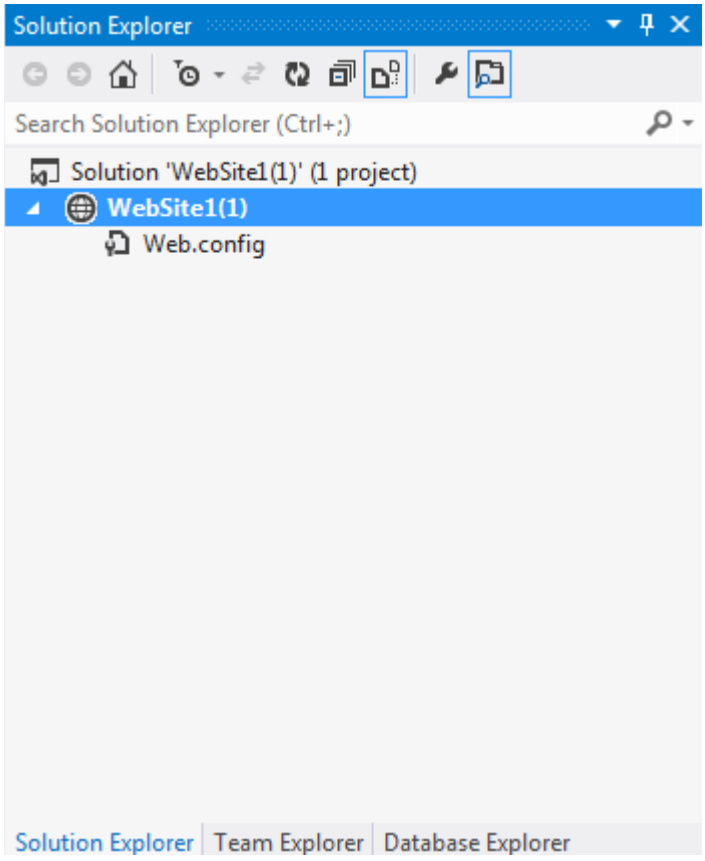


Figure 2c – Creating a New ASP.NET Website: Contents of 'Empty' Website in Solution Explorer

So this is the contents of your website. Pretty empty huh? ☺

All you have to start is a web.config, and that's it. Let's add a web form (ASP.NET page) so we can start doing some stuff.

Right click on your web site location and select Add New Item (Figure 3a) and then select Web Form (Figure 3b). You can leave it as Default.aspx or call it something else.

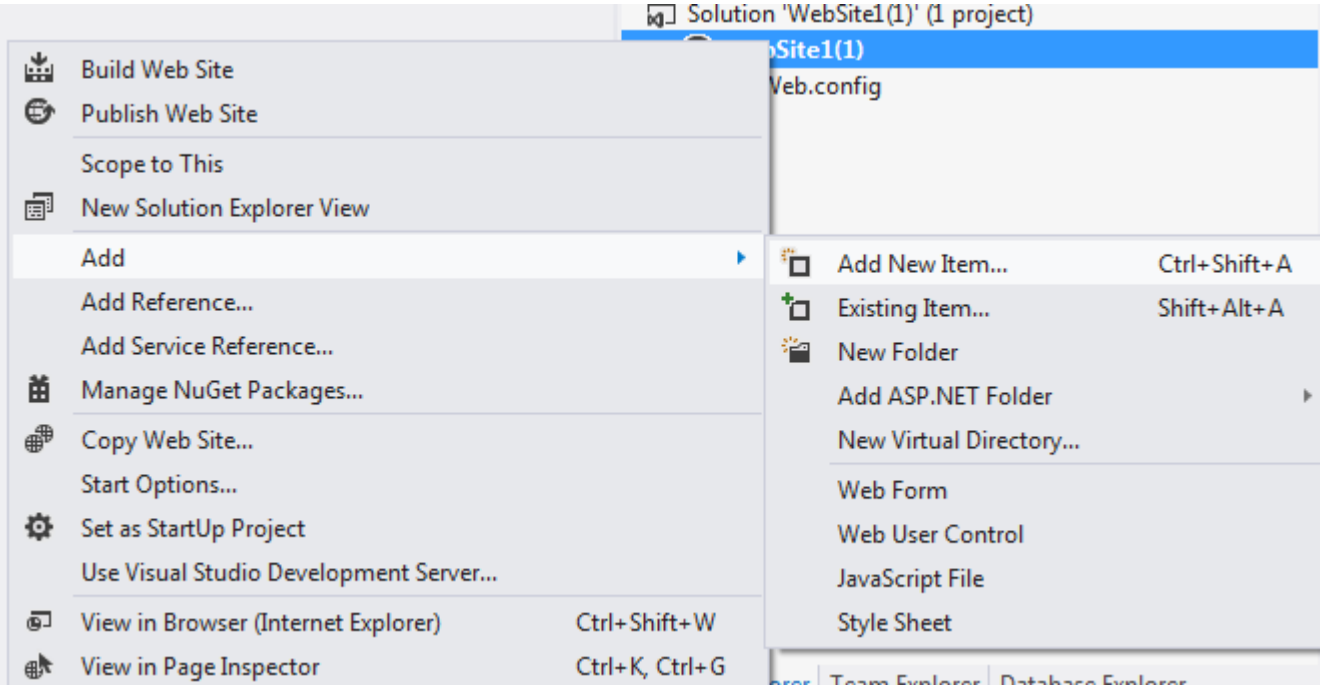


Figure 3a – Adding a Web Form

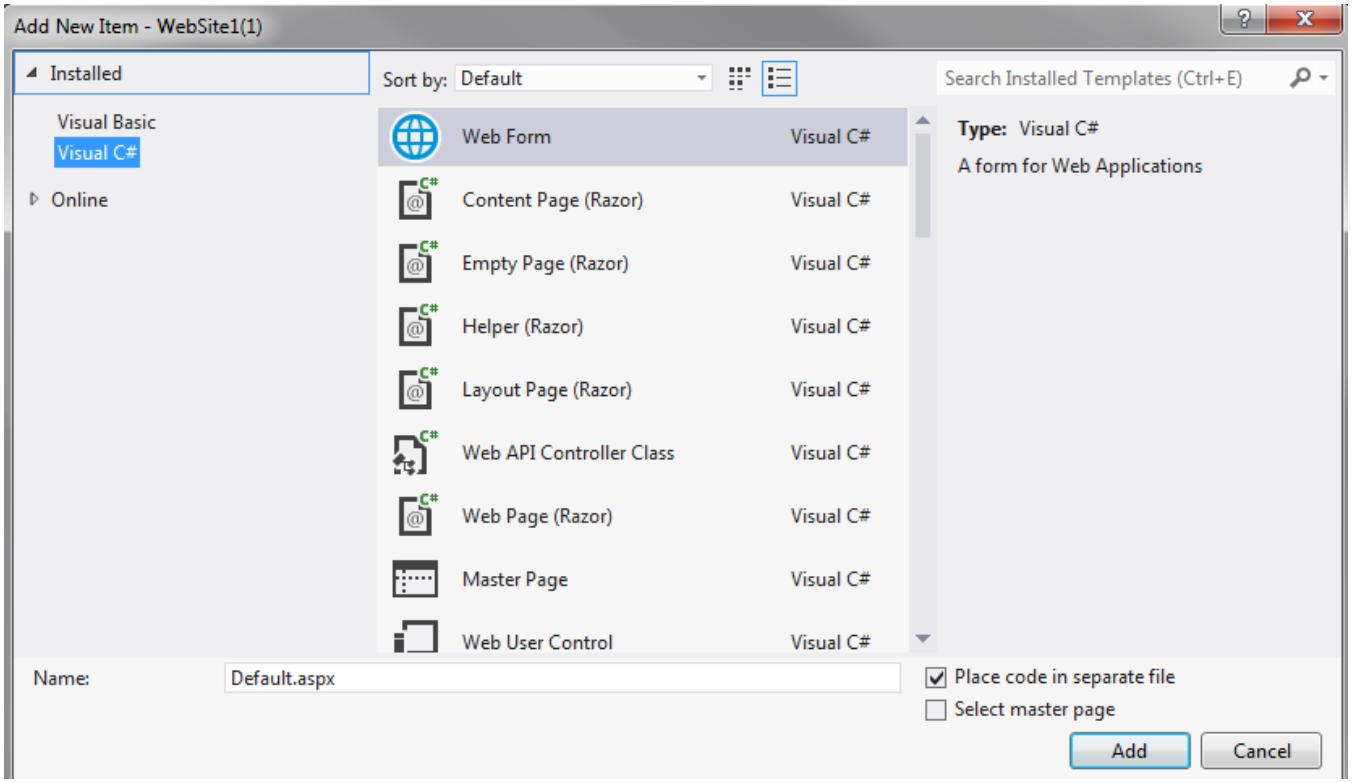


Figure 3b – Adding a Web Form to your Web Site

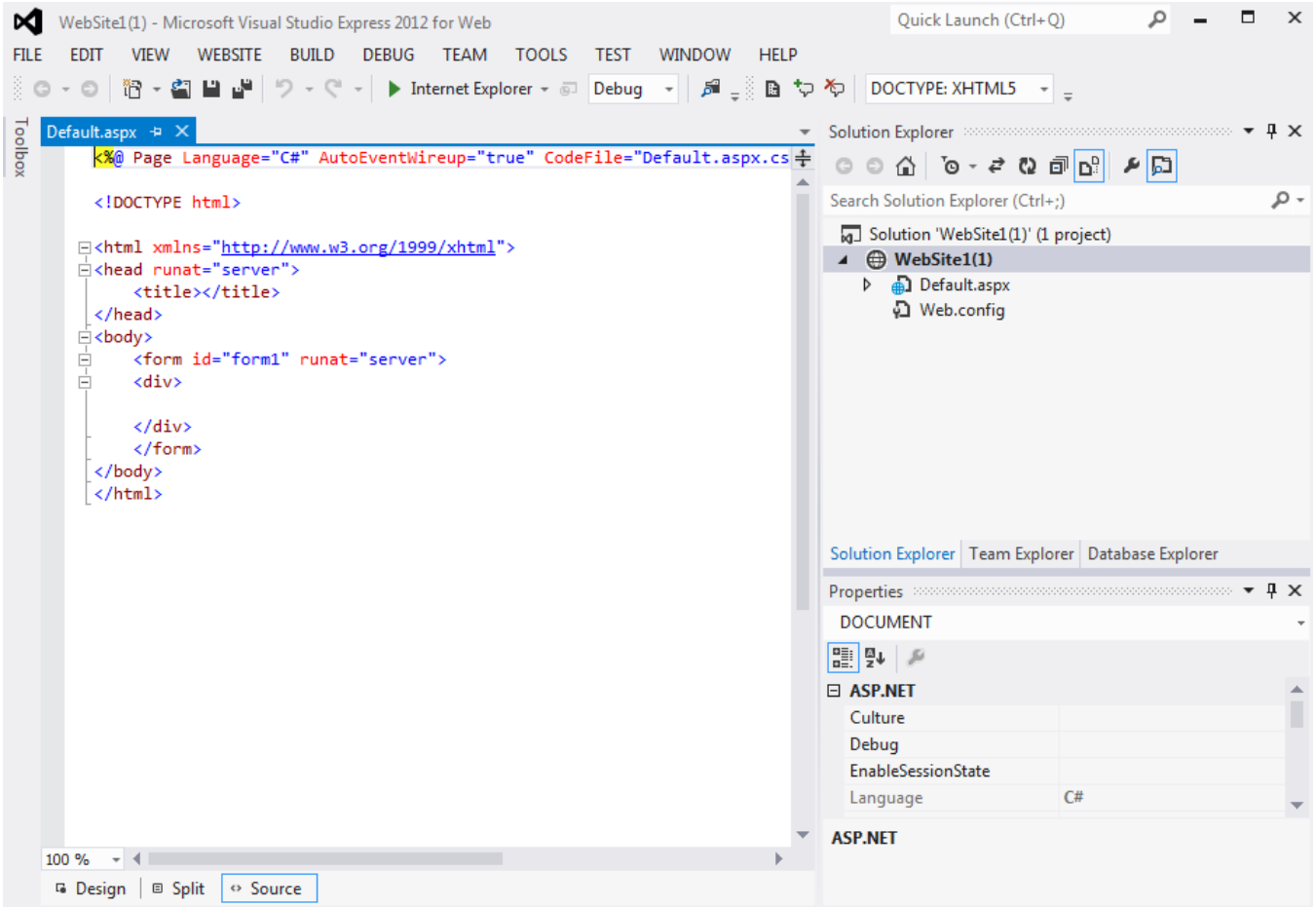


Figure 4 – Workspace of ASP.NET Website

AST.NET Website: Overview

A website (created in Visual Studio) is a collection of resources: static and dynamic web pages, graphic files, style sheets, configuration files, and so on. In addition to various files, a website may contain subdirectories, each of which may contain its own set of files and further subdirectories. A website is akin to a folder on your personal computer: It's a repository for files and subfolders.

Now that we created an ASP.NET website, we have a **web.config** file along with a single ASP.NET page, **default.aspx** (composed of two files: **default.aspx** and **default.aspx.cs**).

Note: Even though I have been talking about websites, files and stuff, a website can have as few as one file or as many as a thousand files.

ASP.NET Website: Reviewing default.aspx

Figure 5 – Reviewing the source HTML of default.aspx

This is a screenshot (Figure 5) of what you will see when you open and view the "Source" of **default.aspx** from the Solution Explorer. When you start adding web controls; the HTML will look a little different.

When reviewing the HTML, it basically looks like any regular HTML page with a few differences.

The top line is a declaration to the ASP.NET engine installed on the web server that it's an ASP.NET page. Then the rest is normal HTML apart from that *runat="server"* tag and attribute.

All server controls must be placed within this tag and it tells the ASP.NET engine to execute the code behind it on web server.

Note: When you add server controls to your page, it automatically gets placed with the *<form runat="server">* tags. But if for some reason it does not, just copy and paste the code.

ASP.NET Website: Reviewing default.aspx.cs

As I mentioned earlier, all your C# code you write will be stored in an **.aspx.cs** file.

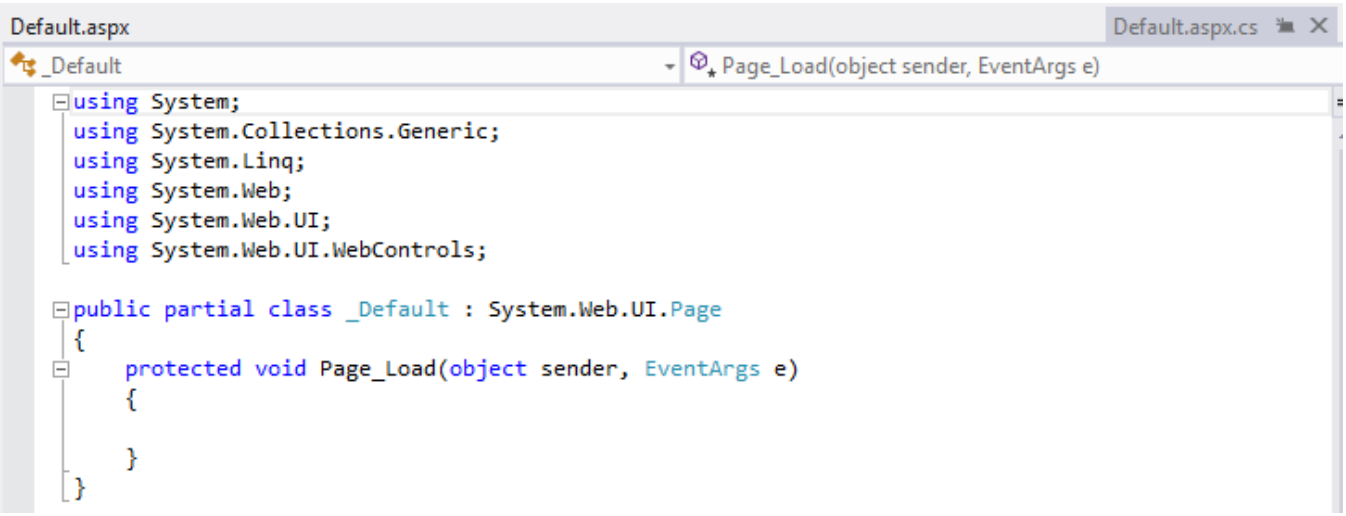


Figure 6 – Reviewing the C# for default.aspx.cs

This page will contain all our C# code that will respond to the different page events as well as any custom methods you create.

The top declarations allow the page to use many of the libraries that are part of the .NET Framework. The code in lines 10 – 13 declares the event for the Page loading (*Page_Load*).

Note: To add line numbers to your code, just go to Tools->Options, General and select the checkbox for Line numbers.

The cool thing about ASP.NET pages is that the .NET Framework is used for all types of application development including Windows applications. This means you can create web pages that can behave like a desktop application on the web.

ASP.NET Website: Adding Some Controls to default.aspx

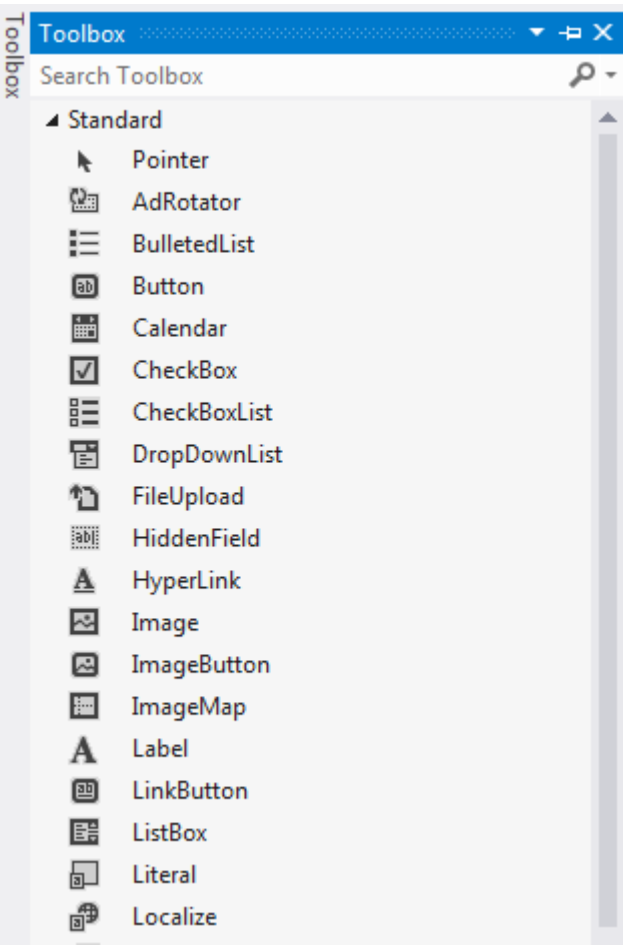


Figure 7 – Toolbox

This screenshot (Figure 7) shows some of the many web controls you can add to an ASP.NET page. The ones we will be using for this example are Label, TextBox and Button.

Note: If you accidentally close the Properties Window, you can re-open it by going to View->Properties Window from the menu.

Drag each one of these controls on the page and try to make it look like the example (Figure 8). The source of the example is in the next screenshot (Figure 9).

Note: The **default.aspx** page needs to be opened in "Design" mode to see how the controls will look on your page. You can be in "Source" mode as well, but it's harder to arrange the controls unless you're very familiar with HTML.

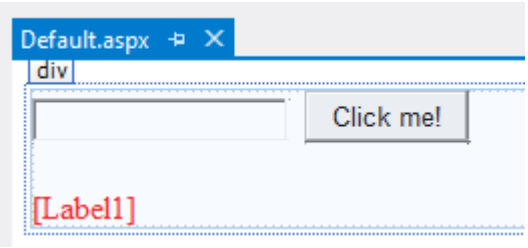


Figure 9– Drag Controls from Toolbox to default.aspx

```
10      <form id="form1" runat="server">
11      <div>
12          <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
13           
14          <asp:Button ID="Button1" runat="server" Text="Click me!" OnClick="Button1_Click" />
15          <br />
16          <br />
17          <asp:Label ID="Label1" runat="server" ForeColor="Red"></asp:Label>
18      </div>
19      </form>
```

Figure 10 – Source of default.aspx after dragging controls

Every time you add a control to your ASP.NET page, once it's compiled on the server, it will create the code for it to render correctly on a web browser.
What that means in English is that the code that you see in your **default.aspx** in Visual Studio, will not be the same code that is sent to the web browser after the ASP.NET engine on the web server renders it.

The ASP.NET engine creates the necessary HTML for the web browser that is requesting the page and sends back what it should look like based upon what controls are on your ASP.NET page.

ASP.NET Website: Adding Some C#

```
11  public partial class _Default : System.Web.UI.Page
12  {
13      protected void Page_Load(object sender, EventArgs e)
14      {
15
16      }
17      protected void Button1_Click(object sender, EventArgs e)
18      {
19          // comment [your personal code notes]
20          // You are setting the value of the label control whenever the
21          // button control is clicked.
22          Label1.Text = "You entered: " + TextBox1.Text;
23      }
24  }
```

Figure 11 – C# in default.aspx.cs

OK, here are some quick steps to replicate this with the least amount of fuss:

1. In the "Design" view of **default.aspx**, double-click on the Button control
2. This will put you in the **default.aspx.cs** page and within the *Button1_Click* code (magically created for you).
3. Type in *Label1.Text = TextBox1.Text*

That's it.

What you did was….

You told Visual Studio that you want code in the Click event of the Button. The line of code you typed in says, "When the Button is clicked, whatever is typed in TextBox1 put it in Label1". The *Text* property of the TextBox and Label are what is visually displayed on the page.

Note: C# is case-sensitive.

ASP.NET Website: Changing Control's Properties

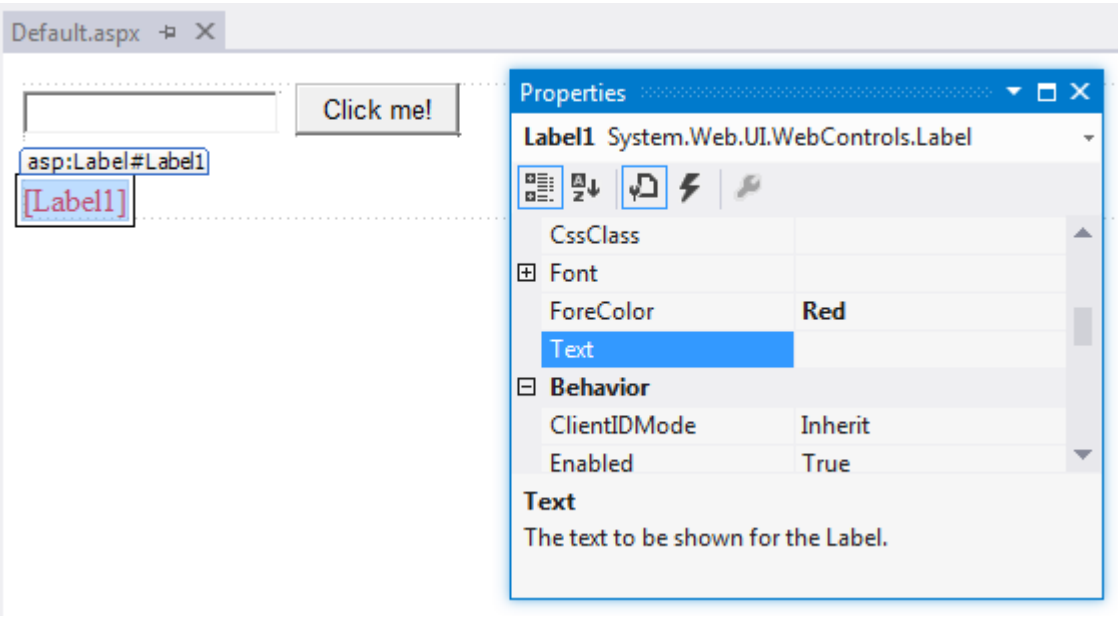


Figure 12 – Properties Window of Label1 control

Every control has properties. A property is like an adjective of something and is usually visual: color, size, text, width, height and style.

In this example, I want to remove the default text of Label1.

I selected my label control on **default.aspx** and the Properties Window displayed the specific properties for the web control. Every time you select a control on your page, the Properties Window automatically changes to display that controls specific properties.

The *Text* property for a label control is what gets displayed on the page when it is visible (you can toggle the *Visible* property too). I want the label to display nothing until the Button is clicked (Button1) and then that one line of code will get executed.

Note: By default, a Label control's *Text* property is the name itself, so I just removed it.

What happens when you remove it???

Now the Label control on the page has its name in brackets [Label1]. That means its *Text* property is empty.

Although it does makes sense if you think about it….

The only way to visually see a label control on an ASP.NET page (in "Design" mode) is by its *Text* property, so how else would you see it?

Visual Studio puts it in brackets so you know its empty, but you can still see it to re-select it and change its properties if needed. Wow, those Microsoft people are pretty smart, huh?

Let's test out our finished example….

ASP.NET Website: Testing our Website

To test our website, you can just right-click the **default.aspx** page in the Solution Explorer and select "View in Browser" (Figure 13).

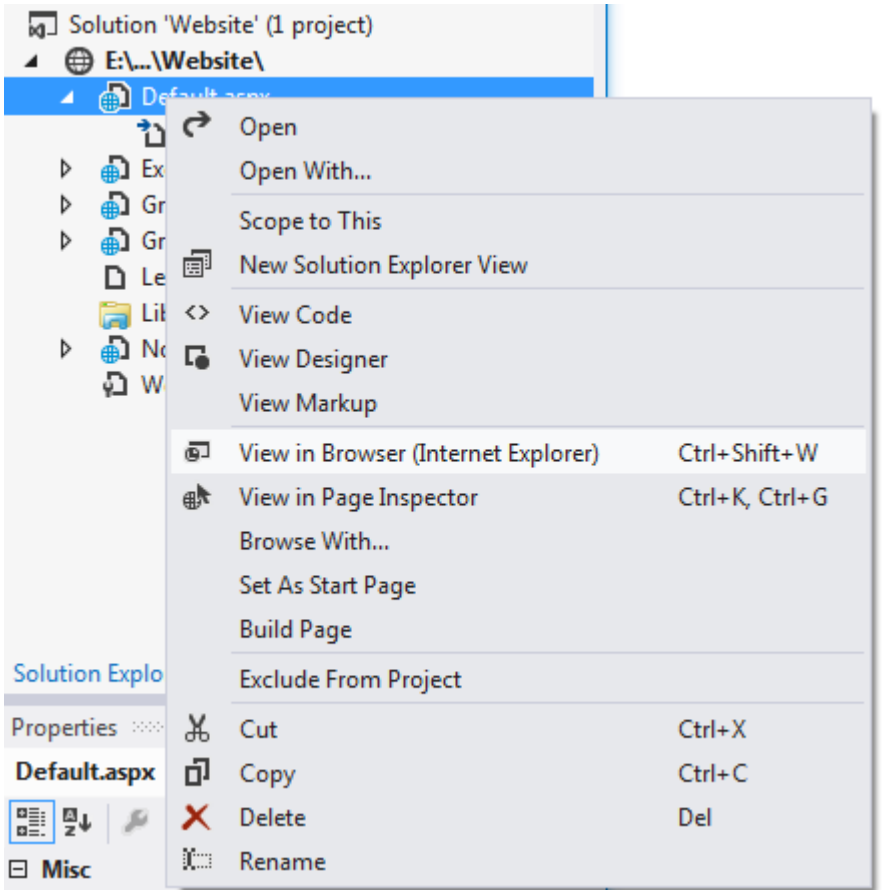


Figure 13 – Testing default.aspx in a web browser

This will open your default web browser and allow you to view your web page through a web server the Visual Studio provides. This temporary web server will show an icon in the taskbar.

Once you close the web browser, after you're done testing, you can let the web server continue to run or close it each time you test a new ASP.NET page.

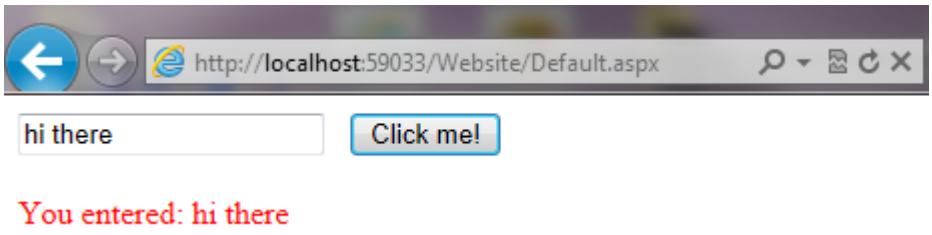


Figure 14 – Viewing default.aspx in a web browser

You can test your first ASP.NET locally and see how it works.

ASP.NET Application: Definition

It's sometimes difficult to define exactly what a web application is. Unlike traditional desktop programs (which users start by running a stand-alone EXE file), ASP.NET applications are almost always divided into multiple web pages. This division means a user can enter an ASP.NET application at different points or follow a link from the application to another part of the website or another web server.

So, does it make sense to consider a website as an application?

In ASP.NET, the answer is **yes**. Every ASP.NET application shares a common set of resources and configuration settings. Web pages from other ASP.NET applications don't share these resources, even if they're on the same web server.

Technically speaking, every ASP.NET application is executed inside an application domain. Application domains are isolated areas in memory; they ensure that even if one web application causes a fatal error, it's unlikely to affect any other application that is currently running on the same computer.

The standard definition for an ASP.NET application is: a combination of files, pages, handlers, modules, and executable code that can be invoked from a virtual directory on a web server.

ASP.NET Basics: Server Controls

As I talked about in the first lesson, ASP.NET has server controls that you can program on the server. There are two types of server controls: HTML and Web controls.

HTML server controls: These are server-based equivalents for standard HTML elements. These are probably not used that much but are good if you need to migrate existing ASP pages to ASP.NET.

Web controls: These are similar to HTML server controls, but they provide a richer object model with a variety of properties for style and formatting details. They also provide more events and more closely resemble the controls used for Windows development. Web controls also feature some user interface elements that have no direct HTML equivalent, such as the GridView, Calendar, and validation controls.

ASP.NET Basics: GradeCalc Application

Here's a simple application to start you off building a simple ASP.NET application.

```
public partial class Default : System.Web.UI.Page
{

    int[] testScores = new int[3];

    protected void Page_Load(object sender, EventArgs e)
    {

        initGrades();
        displayGrade(calcGrades());

    }

    void initGrades()
    {

        testScores[0] = 90;
        testScores[1] = 85;
        testScores[2] = 78;
    }

    float calcGrades()
```

```
    {
        float grade;
        grade = (float)(testScores[0] + testScores[1] + testScores[2]) / 3;
        return grade;
    }
        void displayGrade(float grade)
    {
        if (grade > 90) {
            Response.Write("Score: " + grade + " <br/> Well done!");
        }
        else if (grade > 80)
        {
            Response.Write("Score: " + grade + " <br/> Nice one.");
        }
        else
        {
            Response.Write("Score: " + grade);
        }
    }
}
```

When you look at the **GradeCalc.aspx** page in “Design” view with Visual Studio, what do you see?

Nothing.

That’s because it uses no server controls and the results are being displayed in the *Page_Load* event of the page with a simple *Response.Write* command.

Let’s first document the changes we will make:

UI (User Interface) Controls:

- Four TextBoxes controls; three for each score and one for name.
- One Label control; this will display the score.
- One Button control; this will execute the function(s) that will retrieve the scores, calculate the score and display the score.

Process:

1. The user will enter their three scores in the three TextBoxes.
2. The user will then click the Button.
3. The Button will execute the function(s).
4. A score (average) will be displayed.

Step 1: Setup our web controls

This step will involve grabbing our web controls from the Toolbox (View->Toolbox) and dragging them to the page ("Design" view). We will also change a few properties of the controls to make them a little more descriptive.

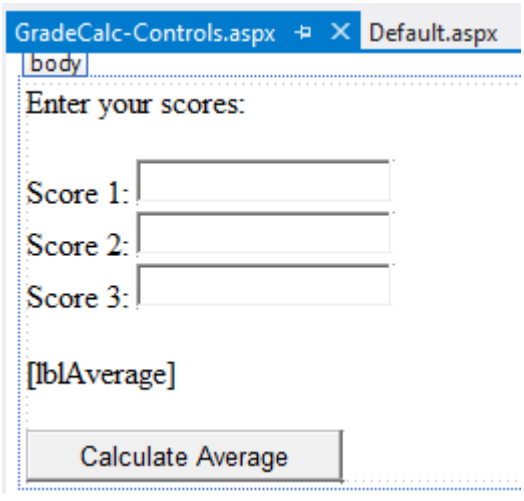


Figure 15 – Designing our new UI – "Design" view

```
10      <form id="form1" runat="server">
11      <div>
12
13          Enter your scores:<br /><br />
14          Score 1:
15          <asp:TextBox ID="txtScore1" runat="server"></asp:TextBox>
16          <br />
17          Score 2:
18          <asp:TextBox ID="txtScore2" runat="server"></asp:TextBox>
19          <br />
20          Score 3:
21          <asp:TextBox ID="txtScore3" runat="server"></asp:TextBox>
22          <br /><br />
23
24          <asp:Label ID="lblAverage" runat="server"></asp:Label>
25          <br /><br />
26
27          <asp:Button ID="btnCalculate" runat="server"
28          Text="Calculate Average" OnClick="btnCalculate_Click" />
29
30      </div>
31      </form>
```

Figure 16 – Designing our new UI – "Source" view

As you look over the both screenshots, it seems to be pretty straightforward from the explanation of what we want our UI to look like.

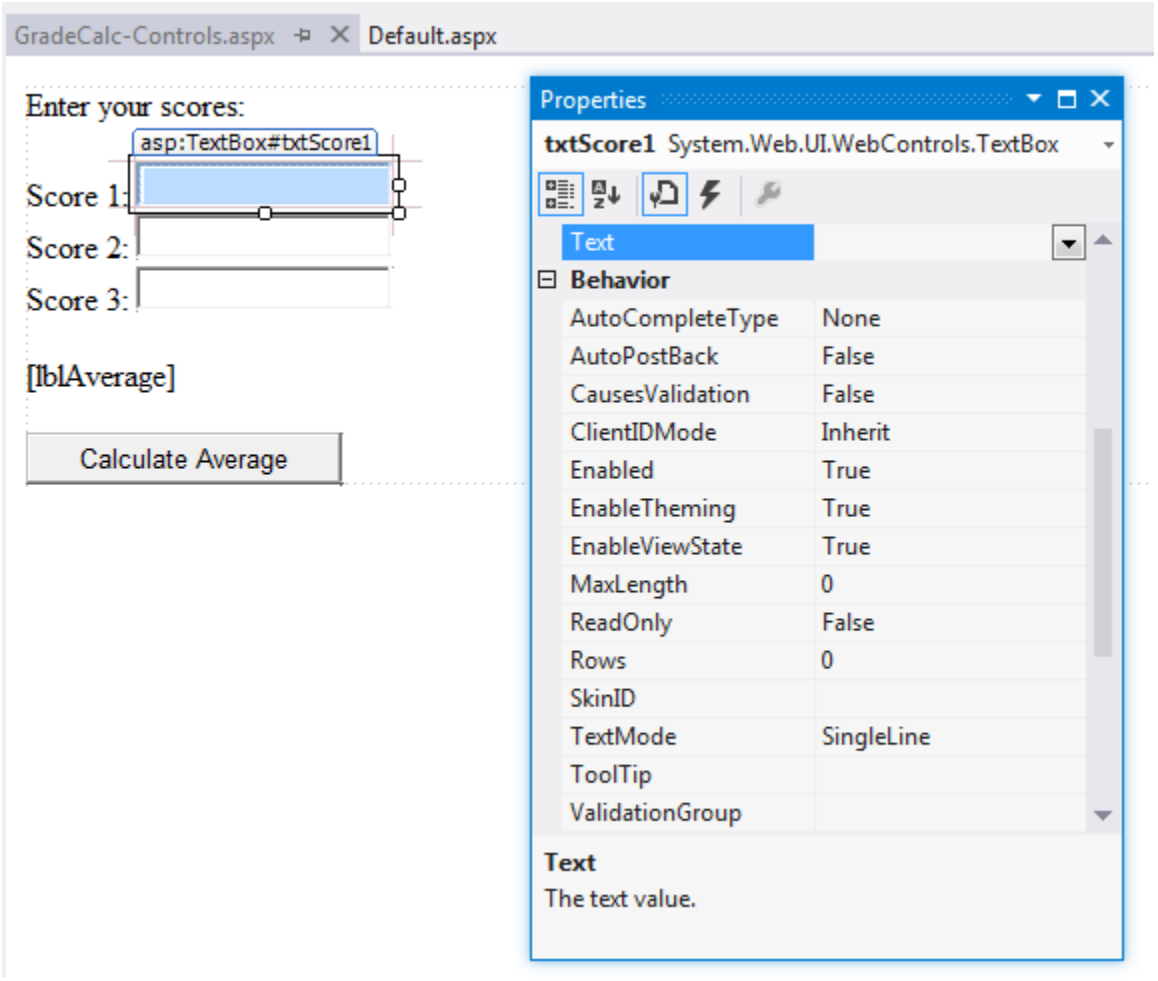


Figure 17 – Changing properties of a web control

Control	Property	Original	New
TextBox	ID	TextBox1	txtScore1
	ID	TextBox2	txtScore2
	ID	TextBox3	txtScore3
Label	ID	Label1	lblAverage
	Text	Label	[blank]
Button	ID	Button1	btnCalculate
	Text	Button1	Calculate Average

Figure 18 – Control Properties that were changed

Changing properties is pretty straightforward:

By first selecting the web control and then going to the Properties Window (View -> Properties Window) you can change any property you want. The table above (Figure 19) shows you every property I changed and what the original property was as well.

By default, Label controls have a Text property of "Label". I want the value of that property to be dynamically displayed in code not as a static text value.

Note: I didn't use a Label control to display the descriptions for the TextBoxes. Personally, it's not really necessary to use a server control unless it's going to be manipulated dynamically in code.

Step 2: Move Methods out of the Page_Load event

Now that we have our web controls ready to go, let's look at our code.

Where is all our code implemented (executed) from? *Page_Load* event

This needs to be changed because now we want everything to happen AFTER we click the Button that says "Calculate Average". This would require the code to be placed in the *Click* event of the button (Figure 19). So just a little cut and paste into the Click event of btnCalculate.

```
67      protected void btnCalculate_Click(object sender, EventArgs e)
68      {
69
70          // calls the initGrades method
71          initGrades();
72
73          // calls displayGrade and passes in parameter from return value of calcGrades
74          displayGrade(calcGrades());
75
76      }
```

Figure 19 – Executing our methods from the Click event of the Button

To get to the *Click* event of the Button (btnCalculate), just double-click on it in the "Design" view and Visual Web Developer will automatically take you to the "code-behind" file and create the C# code for that event. If the code has already been created, Visual Web Developer will just take you to the event.

Note: When you double-click any web control, you will create an event declaration in the "code-behind" file of your web page. The event will be the default event for the particular control selected. Usually, it will be the Click event, but it sometimes is a different one.

Step 3: Modify the initGrades() method

Originally, we "hard-coded" the three scores and put them into the array. Now we have to get them from the three textboxes the visitor will enter. Plus, we also have to convert them to integers first because values stored in textboxes are strings by default.

Since the array is an array of integer values, we need to do a little conversion first.

```
25      // grabs the values from the texboxes and converts them to integers
26      void initGrades()
27      {
28
29          testScores[0] = int.Parse(txtScore1.Text);
30          testScores[1] = int.Parse(txtScore2.Text);
31          testScores[2] = int.Parse(txtScore3.Text);
32
33      }
```

Figure 20 – Changing initGrades() to grab data from textboxes

I used the *Text* property of the textboxes and used the *int.Parse()* method to convert the value entered to an integer.

Step 4: Modify the displayGrade(float) method

The only other change is to display our message in our Label control (lblAverage). This just needs a little change to set the value to the Text property (Figure 22).

```
47      // displays message (no return value but has one parameter)
48      void displayGrade(float grade)
49      {
50
51          // checks conditions (expressions); if true executes code and exits
52          if (grade > 90) { // is grade greater than 90?
53              lblAverage.Text = "Score: " + grade + " <br/> Well done!";
54          }
55          else if (grade > 80) // is grade greater than 80?
56          {
57              lblAverage.Text = "Score: " + grade + " <br/> Nice one.";
58          }
59          else // if first two expressions are not true, just do this
60          {
61              lblAverage.Text = "Score: " + grade;
62          }
63      }
```

Figure 21 – Modifying to display message in label control

Step 5: Finished Version: GradeCalc-Controls.aspx

This should be a good starting point on how to use some basic web controls and incorporate some simple C# code within an event.

Note: If you notice that if you click the submit button without entering the proper values in the text boxes (numbers), you will get an error message. Don't worry, I will talk about this and what you can do to prevent certain type of errors.

Debugging with Visual Studio Express 2012

When creating web sites/applications you are bound to have something happen you didn't want happen or you want to have something happen but it didn't.

This is the case for any kind of web development no matter what the technology or tool you are using. You are never going to get it right the first time. There will always be a little (or a lot) of trial and error no matter how much you plan.

So Visual Studio has something called single-step debugging which allows you to test your assumptions about how your code works and see what's really happening under the hood of you application.

All you do is add a breakpoint to a line of code and a red circle (breakpoint) will appear (Figure 22).

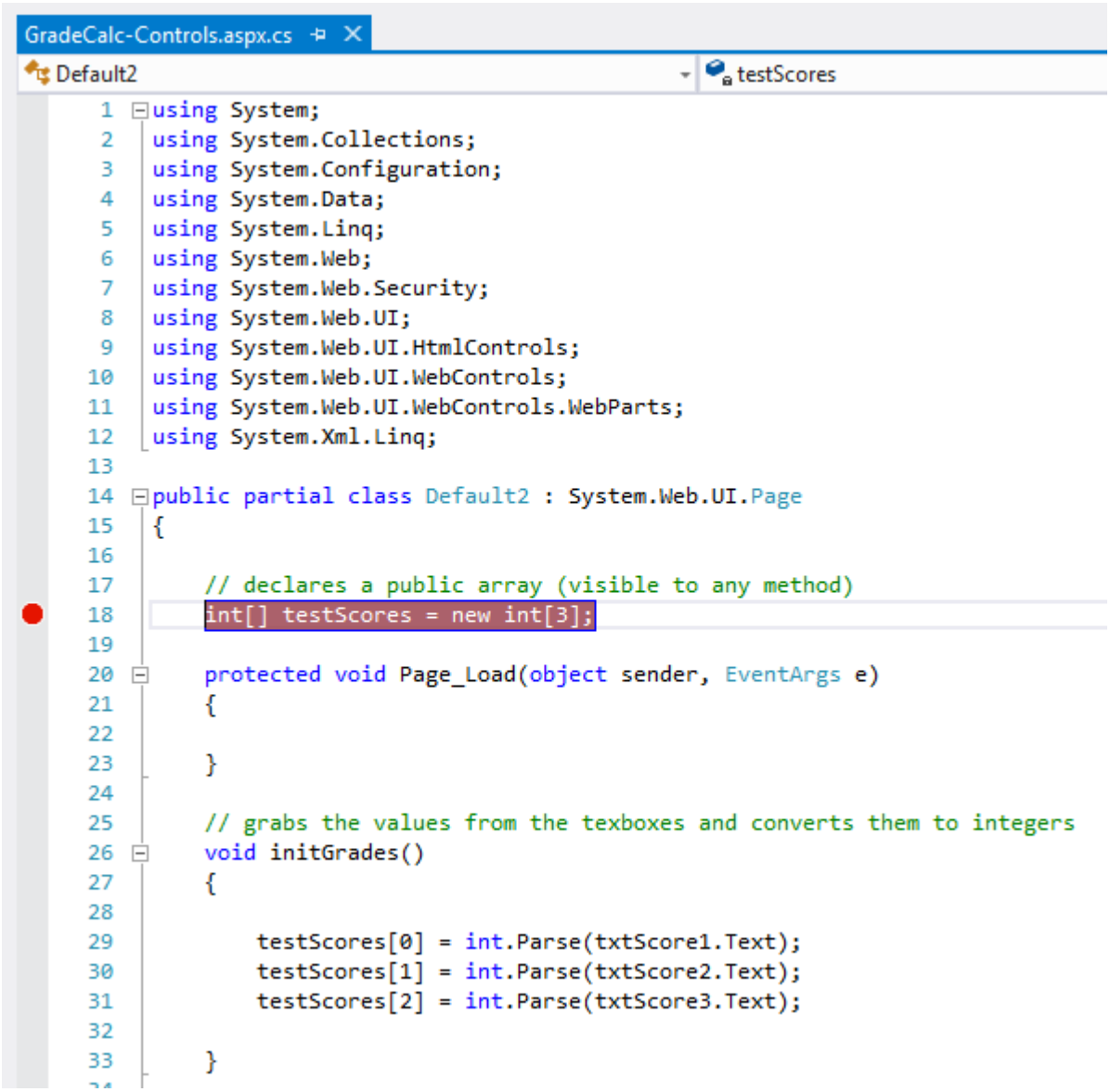


Figure 22 – Setting a breakpoint

This breakpoint will stop execution of your code (when you're debugging your web page) and switch to the code view of Visual Studio and pause at the break point awaiting instructions. At this point you have several options. You can execute the current line by pressing F11. The following line in your code will be highlighted with a yellow arrow, indicating that this is the next line that will be executed (Figure 23).

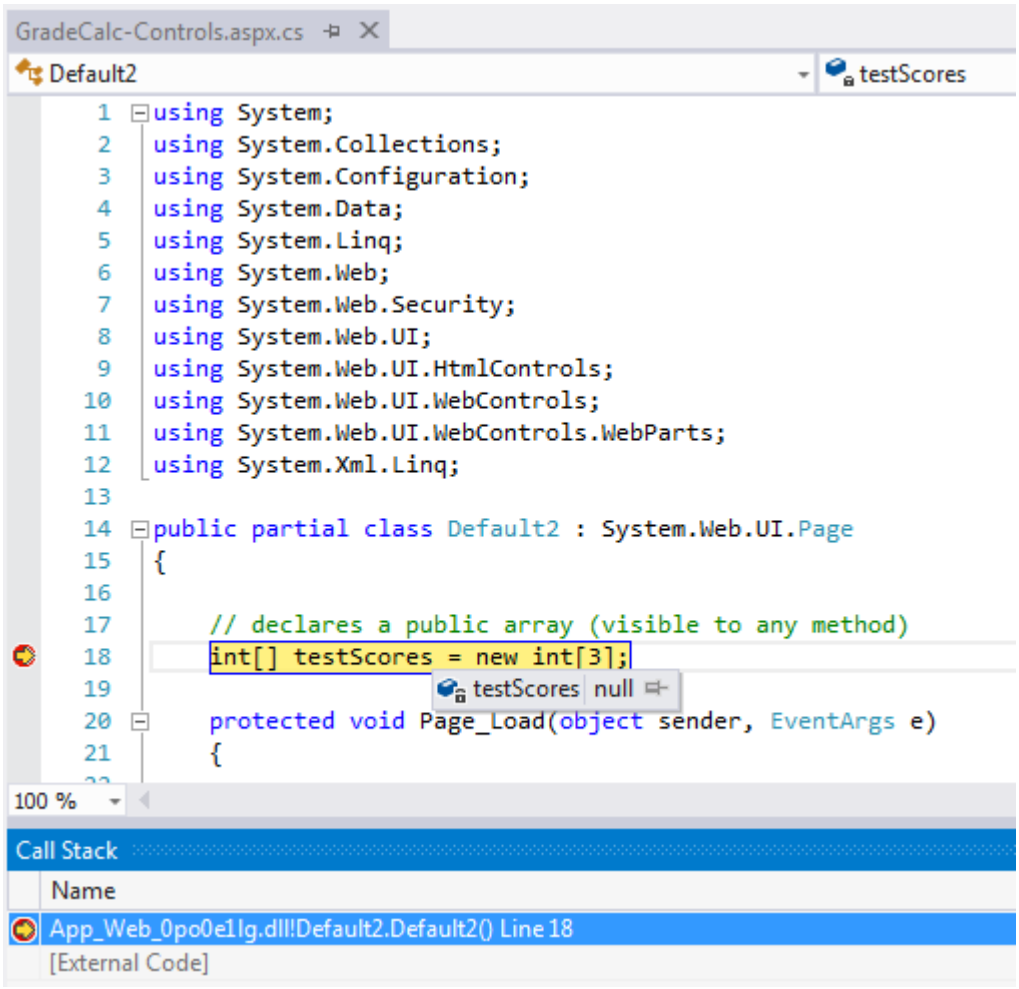
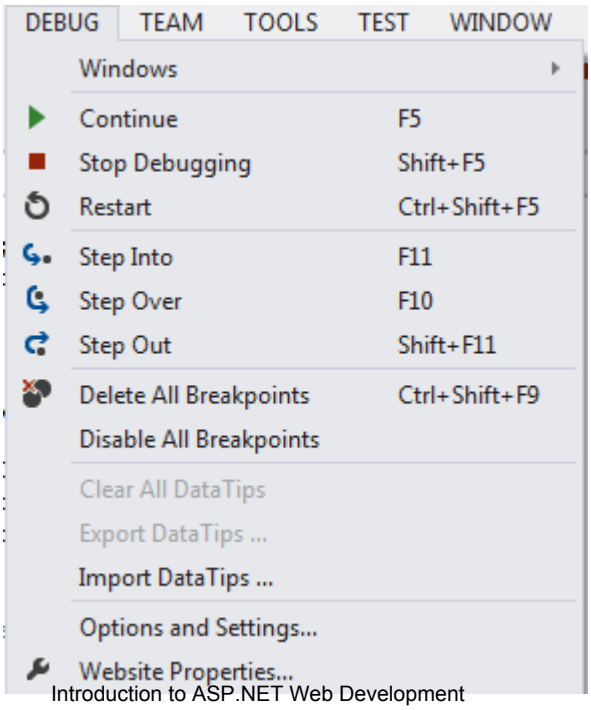


Figure 23 – Execution paused at breakpoint



Notice that while the execution is paused you can place your cursor over the variable, properties, etc and see their current value at that point in time.

You can continue like this through your program; running one line at a time by pressing F11 and following the code's path of execution.

You can use the commands in Figure 24 while in "break" mode.

Figure 24 – Execution paused at breakpoint

Step Over (F10) - The same as Step Into, except it runs methods as though they are a single line. If you selected Step Over while in a method call is highlighted, the entire method will be executed. Execution will be paused at the next executable statement in the current method.

Continue (F5) – Resumes the program and continues to run it normally, without pausing until another breakpoint is reached.

When debugging a large website, you might place a breakpoint in different places of your code and in multiple web pages.

Exception Handling

Most .NET languages support *structures exception handling*. Essentially, when an error occurs in your application, the .NET Framework creates an exception object that represents the problem. You can catch this object using an exception handler. If you fail to use an exception handler, your code will be aborted, and the user will see an ugly error page, which is something you should try and avoid.

The Exception Class

Every exception class derives from the base class System.Exception. The .NET Framework is full of predefined exception classes, such as NullReferenceException, IOException, SqlException, and so on. The Exception class includes the essential functionality for identifying any type of error.

When you catch an exception in an ASP.NET page, it won't be an instance of the generic System.Exception class. Instead, it will be an object that represents a specific type of error. This object will be based on one of the many classes that inherit from System.Exception.

These include diverse classes such as DivideByZeroException, ArithmeticException, IOException, SecurityException, and many more. Some of these classes provide additional details about the error in additional properties.

Structured Error Handling: Exceptions

To use structured exception handling, you wrap potentially problematic codes in the special block structure shown here:

```
try
{
    // Risky code goes here
}
catch
{
    // An error has been detected. You can deal with it here.
}
finally
{
    // Time to clean up, regardless of whether or not there was an error.
}
```

The try statement enables error handling. Any exceptions that occur in the following lines can be "caught" automatically. The code in the catch block will be executed when an error is detected. Either way, whether a bug occurs or not, the final section of the code will be executed last. This can allow for some basic cleanup such as closing a database connection, though we can omit that section from this class.

The act of catching an exception neutralizes it. If all you want to do is render a specific error harmless, you don't ever need to add any code in the catch block of your error handler. Usually, this portion of the code will be used to report the error to the user or log it for future references.

Exception blocks work a little like conditional code. As soon as a matching exception handler is found, the appropriate catch code is invoked. Therefore, you must organize your catch statements from most specific to least specific:

Ending with a catch statement for the base Exception class is often a good idea to make sure no errors slip through.

Exception Handling in Action

Look at this example: NoExceptionHandling.aspx

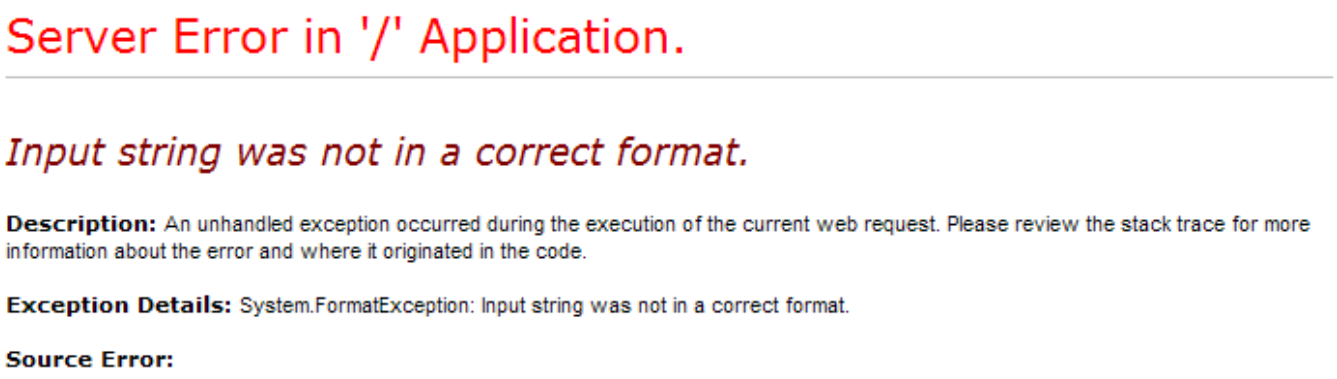
It's using a simple arithmetic of multiplying two integers together:

```
int a = Convert.ToInt16(txtNumber1.Text);
int b = Convert.ToInt16(txtNumber2.Text);
int c = a * b;

lblAnswer.Text = Convert.ToString(c);
```

Works fine as long as you enter integer values in the text boxes, right? But what if you enter a string value or nothing or a decimal?

You'll get something like this:



You at least need to handle the general exception which would cover any type of exception:

```
try
{
      int a = Convert.ToInt16(txtNumber1.Text);
      int b = Convert.ToInt16(txtNumber2.Text);
      int c = a * b;

      lblAnswer.Text = Convert.ToString(c);
}
catch (Exception err)
{
      lblAnswer.Text = "0";
      lblError.Text = "<strong>Error</strong>: " + err.Message;
}
```

So the results will look a little bit nicer to the user.

Of course you could check for other types of exceptions and/or give the user more information on how to correct the error next time, etc.

Nevertheless, you should at the very minimum have basic exception handling in your code whenever you think you could get an unexpected error. More advanced techniques would be logging every error that has occurred in your web application to a log file of some sort so developers can review.