Brief Overview of an ASP.NET Web Application

Contents

[1 Overview 1](#_Toc127788080)

[2 Example 2](#_Toc127788081)

[2.1 Page Source 2](#_Toc127788082)

[2.2 Rendered Page 2](#_Toc127788083)

[2.3 Code-Behind 3](#_Toc127788084)

[3 Page Life Cycle - Abbreviated 4](#_Toc127788085)

[4 C# - Server-side Language 4](#_Toc127788086)

[4.1 String.Format 4](#_Toc127788087)

[4.2 List Class 5](#_Toc127788088)

[5 Visual Studio IDE 6](#_Toc127788089)

[6 Expectations 6](#_Toc127788090)

[Appendix 1 n/a 6](#_Toc127788091)

# Overview

1. As we stated on the first day of class, in general, a web application is a set of web pages composed of HTML with server-side code embedded. When a web page is requested, the server:
2. Loads the web page into memory
3. Parses the file line-by-line
	* HTML is sent to an output buffer
	* Server-side code is stripped out and executed. The output of this code is sent to the output buffer.
4. The output buffer is sent to the browser.

In this class, we will create ASP.NET web applications. Next, we explain how the process above works in ASP.NET.

1. An *ASP.NET web application* is one or more *web forms* (this is the technical name, though we will frequently just call them *pages* or *web pages*)*.* A web form:
* Is a text file which contains HTML and *server controls* (also called: *web controls* or *controls*). Server controls use *ASP.NET* tags to markup UI controls. Server controls can be manipulated by code on the server, and ultimately render as HTML.
* Has an *.aspx* extension (as opposed to *.html*)
* Has an associated *code-behind* file that contains event handlers and other server-side code. The code-behind file as an *.aspx.cs* extension where “cs” denotes the server-side language we will use, *C#*. A few other languages: Visual Basic (.vb), C/C++ (*.cpp*), Python (*.py*), Go (*.go*),

# Example

## Page Source

Consider the *HelloWorld.aspx* web form below. We will explain some of the new elements in class.

<%@ Page Language="C#" AutoEventWireup="true"

 CodeBehind="HelloWorld.aspx.cs" Inherits="aspbasics1.pages.HelloWorld" %>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

 <title>Hello World</title>

</head>

<body>

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

 <div>

 <p>Name:

 <asp:TextBox ID="txtName" runat="server"></asp:TextBox>&nbsp;

 <asp:Button ID="btnHello" runat="server" Text="Button"

 OnClick="btnHello\_Click" />&nbsp;

 </p>

 <p>

 <asp:TextBox ID="txtMessage" runat="server" Height="74px"

 TextMode="MultiLine" Width="288px"></asp:TextBox>

</p>

 </div>

 </form>

</body>

</html>

A *server control* represents a server-side object that is programmable and is used for displaying information on a web page, collecting information from a user, or allowing a user to cause some action by interacting with it. For example:

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

## Rendered Page

When a web form is delivered to a browser, it is converted to HTML. For example:

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>Hello World</title>

</head>

<body>

 ...

<form method="post" action="./HelloWorld.aspx" id="form1">

<div>

<p>Name:

<input name="txtName" type="text" id="txtName" />&nbsp;

|  |
| --- |
| On the server, the tag above was:<asp:TextBox ID="txtName" runat="server"></asp:TextBox>&nbsp; |

<input type="submit" name="btnHello" value="Button" id="btnHello" />&nbsp;

|  |
| --- |
| On the server, the tag above was:<asp:Button ID="btnHello" runat="server" Text="Button" OnClick="btnHello\_Click" />&nbsp; |

</p>

<p>

<textarea name="txtMessage" rows="2" cols="20" id="txtMessage"

 style="height:74px;width:288px;">

in Page\_Load

Dave, Hello

</textarea>

|  |
| --- |
| On the server, the tag above was:<asp:TextBox ID="txtMessage" runat="server" Height="74px" TextMode="MultiLine" Width="288px"></asp:TextBox> |

</p>

</div>

</form>

</body>

</html>

## Code-Behind

Every web form has a *code-behind* file which is a class holds the event handlers associated with the page. For example, the code-behind file for *HelloWorld.aspx* is: *HelloWorld.aspx.cs*:

namespace aspbasics1.pages {

 public partial class HelloWorld : System.Web.UI.Page {

 protected void Page\_Load(object sender, EventArgs e) {

 txtMessage.Text = "in Page\_Load" + Environment.NewLine;

 }

 protected void btnHello\_Click(object sender, EventArgs e) {

 string msg = txtName.Text + ", Hello";

 txtMessage.Text += msg;

 }

 }

}

Notice that the class above, *HelloWorld* is a *partial class*. The other part of the class defines the UI elements (the server controls). They are defined in a file named: *HelloWorld.aspx.designer.cs*. We generally don’t edit this code by hand. When we use VS’s visual designer to build a page, the designer modifies this file. It looks something like this:

namespace aspbasics1.pages {

 public partial class HelloWorld {

 protected global::System.Web.UI.HtmlControls.HtmlForm form1;

 protected global::System.Web.UI.WebControls.TextBox txtName;

 protected global::System.Web.UI.WebControls.Button btnHello;

 protected global::System.Web.UI.WebControls.TextBox txtMessage;

 }

}

# Page Life Cycle - Abbreviated

When a user is viewing the page on a browser, when the button is pressed, the following occurs:

* The page is posted back to the server.
* An instance of this class is created.
* Instances of all server controls are created.
* The page is *run*:
	+ The *Page\_Load* event handler is executed.
	+ The *btnHello\_Click* event handler is executed
* The results are translated to HTML and sent to the browser.

# C# - Server-side Language

The code is written in the C# language. The basic features of the C# language (variables, loops, if, *etc.*) are nearly identical to Java: [Data Types](https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2008/ms228360%28v%3Dvs.90%29), [Flow Control](https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2008/ms228393%28v%3Dvs.90%29), [Looping Statements](https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2008/ms228598%28v%3Dvs.90%29), [Arrays](https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2008/ms228389%28v%3Dvs.90%29), [Methods](https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2008/ms228506%28v%3Dvs.90%29)C(You write helper methods exactly as you would in Java).

MSDN: [C# Ref](https://msdn.microsoft.com/en-us/library/618ayhy6.aspx), [C# Prog Guide](https://msdn.microsoft.com/en-us/library/67ef8sbd.aspx)

Some [API Classes](https://learn.microsoft.com/en-us/dotnet/api/?view=net-7.0):

* [String](https://docs.microsoft.com/en-us/dotnet/api/system.string?redirectedfrom=MSDN&view=netframework-4.7.2), String.Format: [SteveX](http://blog.stevex.net/string-formatting-in-csharp/), [C# Corner](https://www.c-sharpcorner.com/UploadFile/mahesh/format-string-in-C-Sharp/), [MSDN](https://docs.microsoft.com/en-us/dotnet/api/system.string.format?view=netframework-4.7.2) (See my example, *formattingStrings.aspx* in code download)
* [Math](https://docs.microsoft.com/en-us/dotnet/api/system.math?redirectedfrom=MSDN&view=netframework-4.7.2)
* [Convert](https://docs.microsoft.com/en-us/dotnet/api/system.convert?redirectedfrom=MSDN&view=netframework-4.7.2)
* [List](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?redirectedfrom=MSDN&view=netframework-4.7.2) (See my example, *listExample.aspx* in code download)

Comparison of Java & C#. [Microsoft](https://download.microsoft.com/download/D/E/E/DEE91FC0-7AA9-4F6E-9FFA-8658AA0FA080/CSharp%20for%20Java%20Developers%20-%20Cheat%20Sheet.pdf) (large pdf), [Dr. F.McCown](http://www.harding.edu/fmccown/java_csharp_comparison.html), [Wikipedia](http://en.wikipedia.org/wiki/Comparison_of_Java_and_C_Sharp), [Genamics](http://genamics.com/developer/csharp_comparative.htm)

OO Concepts (subclasses, inheritance, interfaces, *etc*)are very similar to Java and will be consider later.

As we can see above in the click event, we can extract the value in the text box (*txtName*), and put a value in the text area (*txtMessage*). The [*TextBox*](https://docs.microsoft.com/en-us/dotnet/api/system.web.ui.webcontrols.textbox?view=netframework-4.7.2)class has a rich set of properties which can be set at design time or programmatically. The *TextBox* class is a subclass of [*WebControl*](https://docs.microsoft.com/en-us/dotnet/api/system.web.ui.webcontrols.webcontrol?view=netframework-4.7.2)which has additional properties that are inherited. All these can be seen in the VS Properties window.

## String.Format

See *formattingStrings.aspx* in the code download.

string name = "Alton Reid";

int age = 33;

double weight = 187.438982342;

double salary = 67882.35;

// Example 1

msg = String.Format("-->Name: {0}, Age: {1}, Weight: {2}, Salary: {3}",

 name, age, weight, salary);

-->Name: Alton Reid, Age: 33, Weight: 187.438982342, Salary: 67882.35

// Example 2

msg = String.Format("-->Salary: {0:c}", salary);

-->Salary: $67,882.35

// Example 3

msg = String.Format("-->Weight: {0:0.000}", weight);

-->Weight: 187.439

// Example 5

double x = 889239.92736972621;

msg = String.Format("-->x: {0}, x: {0:0,0.00}", x);

-->x: 889239.927369726, x: 889,239.93

Aligning with Spaces: all values are right-justified by default. Specifying a negative width results in a left-justification

// Example 7

double[] vals1 = { 22.34234, 535.23433, 3.8584 };

double[] vals2 = { 333.543523, 2.343, 66.3283 };

string[] cities = { "New York City", "Chicago", "San Francisco" };

for(int i=0; i<vals1.Length; i++) {

 msg = String.Format("{0,8:0.00}{1,8:0.00} - {2,-15}", vals1[i], vals2[i], cities[i]);

 txtMsg.Text += msg + Environment.NewLine;

}

 22.34 333.54 - New York City

 535.23 2.34 - Chicago

 3.86 66.33 - San Francisco

## List Class

See *listExample.aspx* in the code download.

// The List class is like Java's ArrayList. ToArray turns list into an int array, and the array's

// ToString returns the string representation

List<int> ints = new List<int>() { 1, 2, 3 };

msg += "--> " + string.Join(",", ints.ToArray());

ints.Add(99);

// There is no getter. Instead, C# uses an indexer, which is a special type

// of property that that allows access to the collection using an integer (in this case)

// index and array notation.

int y = ints[2];

// Remove an object

ints.Remove(2);

// Remove at an index

ints.RemoveAt(0);

// Indexed loop

for (int i=0; i<ints.Count; i++) {

 msg += ints[i] + " ";

}

// foreach loop

foreach (int val in ints) {

 msg += val + " ";

}

# Visual Studio IDE

Although we can write web forms by hand in any text editor, we will use Visual Studio to author page as this rapidly speeds up development. You used VS in the two labs associated with this topic.



# Expectations

1. Know the structure of an ASP.NET page.
2. Use the basic web controls (Label, TextBox, DropDownList, ListBox, CheckBoxList, RadioButtonList, Button) to write simple click event handlers for a Button.
3. Know the basic validators (CompareValidator, RangeValidator, RequiredFieldValidator) and how they work.
4. Describe how to do custom validation.

Appendix

1. n/a